L E X W A R E M A N U A L

Computer Programs for Lexicography
Developed at the University of Hawaii

Second Edition, partial draft
PC (MS-DOS) version

Robert Hsu
Linguistics Department
University of Hawaii
Honolulu, Hawaii 96822

H5.

This material is based upon work supported by the National Science Foundation under Grant No. BNS 7924700. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

CHAPTER 1. ORIENTATION

This chapter provides various types of background information on the programs, but no essential information on using them. If you want to begin using the system immediately, begin with Chapter 2. However, you should read this chapter before getting too far along in order to get an idea of what to expect and not to expect of the programs.

1.1 ORIENTATION TO THE SYSTEM

1.1.1 SCOPE OF THE SYSTEM

The system provides 1) a general, flexible, and relatively convenient set of conventions for encoding the potentially complex structure of dictionary entries (see 3.2), and 2) programs for processing data so encoded (see 3.4, 3.5). The most commonly used programs are those for formatting and printing out a dictionary (from tape or disk storage), and for generating a "finderlist" from bilingual dictionaries. A finderlist is an index of the entries in a dictionary by keywords and phrases contained in the definitions (see Appendix I). There are also programs to sort the entries by their headwords according to any specified alphabetical scheme, to re-order the information within each entry, and to sort headwords by various types of information in the entry. Since a static system of programs cannot hope to meet the continually developing and special needs of every user, the system is designed so that new programs, usually very short, can be written by a programmer relatively quickly by calling on the basic functions provided in the system (see Chapter 6 for a technical description).

1.1.2 USAGE

A linguist beginning to use this system typically already possesses a substantial body of lexical data--e.g. in the form of drafts of dictionary entries on file slips or other media. The first step is to design some conventions for representing this data within the overall "band" format (as described in Chapter 2 and Sections 3.2 and 3.3 of Chapter 3), and then to cast some of the data into this format. This initially need not even be done on the computer. As the range of entries expands, the conventions normally also require expansion and adjustment. Programs (particularly the three mentioned at the end of Chapter 2) are run on the data as soon as a small amount has been entered into the computer, and the outputs of the runs (in the form of other files or of hard-copy printouts) are studied carefully with a view to understanding and validating the structural relations that they exhibit. Insights thus gained invariably lead to further revisions in the conventions.

Programs are run periodically as more data is entered into the masterfile, the outputs are studied, and fieldwork is often interspersed with working on the computer. As a result of better understanding of both the data and the system, a user may sometimes decide to radically reorganize the already entered data. This is a normal mode of progress and, burdensome as the reorganization is, it should not be regarded as a set-back.

On no account should one wait until all of the available data is entered before running any of the programs. The usefulness of the system lies not so much in recording lexical data as in helping to discover and explore the regularities--lexical, grammatical, cultural, etc.--present in the data. Obviously, the earlier this process of discovery is begun the better, since it affects the way material is subsequently entered and often requires revision of previously entered material. It is this feed-back that the user gets from the interaction, both smooth and rough, between the data and the system of recording and manipulating it that is the major benefit of using the system. The user, of course, must be willing to experiment.

To make changes in the dictionary the user edits the masterfile, not the files output by programs of the kind mentioned above. However, some programs reorganize the entire dictionary, so that their output actually becomes the new masterfile. An example would be a program that alphabetizes the file, or one that rearranges the material within entries.

Since it is primarily an exploratory tool rather than a passive receptacle for information, effective exploitation of the system depends on the linguist's being not only as thoroughly familiar with the data as possible and actively interested in its structure, but also the same person's being intimately involved in the day-to-day operation of the project: casting material into band format, editing the data, designing and revising conventions, deciding what programs and parameters to use, studying the outputs, etc. Otherwise, as when these tasks are delegated to linguistically less knowledgeable and possibly less interested subordinates, the system is not very useful.

Another significant characteristic of this system is the independence of the data from the programs, which has the consequence that data entered in the "band" format may be processed usefully by other programs. Since the format is extremely simple and generic, it can be manipulated in useful ways by other general-purpose text-processing systems, such as sufficiently flexible editors and text database systems.

1.1.3 ENVIRONMENT OF THE PROGRAMS

The programs are designed to be run in "batch mode" rather than interactively. That is, after being launched a program normally

runs from beginning to end without further intervention from the user. Afterwards the user retrieves and examines the output. While the programs provide no facilities for interaction, interactive facilities elsewhere on the computer may be used to prepare the data, launch the batch job, and examine the output.

The programs, furthermore, handle only "sequential" files. A program typically processes an entire dictionary file, by reading it in a single pass from beginning to end.

The dictionary programs are written in the SNOBOL4 programming language, specifically the version known as SPITBOL. A SPITBOL compiler, as well as a general-purpose sort program, must be available on a computer in order to run these programs. Some version of SPITBOL is available on most models of computers and at most academic computing centers, and a sort utility is normally also available.

The programs were originally developed on IBM mainframes under the OS/360 operating system and its descendants, and have been transported to personal computers running the MS-DOS family of operating systems. It is on these two systems that the programs have the most users, and this is reflected by the existence of the two separate versions of this manual, the IBM mainframe version and the PC (MS-DOS) version. The mainframe version also gives details for installing the programs on a few non-IBM mainframes, on which the programs have also been run. There is a possibility that the programs will become available on the Macintosh in the future.

1.1.4 WHAT THE SYSTEM DOES NOT DO

This system is by no means a complete toolkit for the lexicographer. Furthermore, lexicography being a diverse and wide-ranging collection of pursuits, some practitioners may find little overlap between their needs and what these programs have to offer.

The system does not, for instance, provide a special editor for lexical files. However, many powerful editors are available on most computers.

Nor will users find here the ability to instantly access any part of the lexical file via index terms and other retrieval specifications. The kind of retrieval that these programs provide is not instantaneous. If immediate on-line access is desired, a better solution may be provided by text database and regular database systems, although at a cost in certain kinds of flexibility.

The system is also not oriented to publication by having built-in facilities for producing camera-ready copy for publication. This lack is mitigated somewhat by the ease of development of special-purpose programs within the system. It is thus reasonably

straightforward to write a special program to convert a particular lexical file into a format acceptable by a particular typesetting system. This has in fact been done many times, for different dictionaries and for different typesetting systems, exhibiting a flexibility that might even be unattainable with a built-in facility.

The making of text concordances, one of the major tools of lexicography, is beyond the original purpose of these programs, which were designed to handle lexical files rather than text. However, they could easily be turned to such a use.

The dictionaries that these programs handle are not in a suitable form for use as the lexical component of a computerized grammar of a language, or as a mechanical translation or information retrieval system. The dictionary files are basically computer-stored and explicitly structured versions of the usual kind of human-oriented bilingual or monolingual dictionaries, whether scholarly, pedagogical, or general.

Nor are these programs particularly helpful for comparing several dictionaries of a language, for studying the structure of definitions in a particular dictionary, or for other lexicological pursuits.

The gathering of statistics from the lexicon, and statistical computations, form another area that these programs leave untouched.

The following short bibliography, which is neither complete nor uniform in its coverage, is provided to point the user to other systems that emphasize some of these other areas. For instance, Venezky 1973 describes a system for retrieval of words and their contexts from large corpora of text together with the editing and construction of dictionary entries. The Revard and Olney items illustrate computer-assisted lexicological studies. A complete bibliography of computer-aided lexicography would run several hundred items in length. Bailey (1974) is a draft of part of such a bibliography; Kipfer (1982) is a more recent bibliography. It will be noticed that almost all of the projects reported in the literature deal with monolingual dictionaries of languages with a literate tradition --- English of various epochs, French, Russian, etc., for which the primary logistics problem is the collection and collation of millions of citations.

Bailey Richard W. 'Computer-assisted Lexicography'. American Journal of Computational Linguistics. Vol. II, No. 1 (Finite String), 1974.

Bailey, Richard W. and Jay L. Robinson. 'Computers and Dictionaries', in Angus Cameron, Roberta Frank, and John Leyerle, eds., Computers and Old English Concordances, Toronto 1970. 94-102.

Gellerstam, Martin, ed., Studies in Computer-Aided Lexicology. Stockholm: Almqvist & Wiksell International, 1988.

Goetschalckx, J., and L. Rolling, eds., Lexicography in the Electronic Age. New York and Amsterdam: North Holland, 1982.

Josselson, Harry H. 'Lexicography and the Computer', in To Honor Roman Jakobson: Essays on the Occasion of his Seventieth Birthday. The Hague 1967. II.1046-59.

Kipfer, Barbara Ann, "Bibliography of computer applications in lexicography", in Dictionaries, Journal of the Dictionary Society of North America. #4 (1982), pp.202-237.

McDavid, Raven Jr. and Audrey Duckert eds. 'Lexicography in English'. (Part VI: Technology in Lexicography) Annals of the New York Academy of Science, Vol-211 (June 8, 1973).

Olney, John, Carter Revard, Paul Ziff. Toward the Development of Computational Aids for Obtaining a Formal Semantic Description of English. Santa Monica: Systems Development Corporation, report SP-2766/001/00. 1968.

Revard, Carter. 'On the Computability of Certain Monsters in Noah's Ark'. Computer Studies in the Humanities and Verbal Behavior, 2, 82-90 (1969).

Urdang, L. 'The systems design and devices used to process the Random House Dictionary of the English Language.' Computers and the Humanities, 1, 31-33 (1966).

Venezky, Richard L. 'Computational aids to dictionary compilation', in R. Frank and A. Cameron, eds. A Plan for the Dictionary of Old English. Toronto 1973.

Woelck, Wolfgang. 'A Computerized Dictionary of Andean Languages'. Language Sciences 8 (Dec. 1969).

One might also scan the following journals for relevant material: Computers and the Humanities; Journal of the Association for Computational Linguistics; Journal of the Association for Literary and Linguistic Computing.

1.1.5 HISTORY AND STATUS OF THE SYSTEM

The system documented here began life as separate programs written for the IBM System/360 in the late 1960's. A long series of accretions, and two or three major revisions and consolidations of the entire system, have led to the present system, which continues to undergo expansion and revision. The entire development has been

guided and motivated by the needs of actual dictionary projects--several dozen projects on languages from a wide variety of families (see Appendix III for a sketch of the background of these programs). The basic data structure and program organization have, however, changed little since the beginning. More extensive changes have occurred in the areas of making the disparate programs more internally consistent and of organizing the system to be more easily expandable. If any major changes to the system are made in the future they will be made in such a way as to preserve compatibility with already existing dictionary files.

Since 1985, the programs have seen increasing use on personal computers running the MS-DOS operating system. Minor additions to the PC version of the system have been made, to address the needs and possibilities presented by this environment. Apart from these, however, a policy has been followed of maintaining uniformity of the system across different environments, even if it has meant neglecting special capabilities of particular environments.

The history and circumstances of the development of the programs have given the system some peculiar characteristics. They were originally designed and written not for use by lexicographers themselves, but rather for the programmer's own convenience in responding to requests from the lexicographers. Thus, for instance, programs that were often needed were packaged into a form that was easy for the author to use and adjust for different projects. These programs thereby also became a little easier for the non-programmer to use. Due to increasing demand, some of the programs were then gradually released for non-programmers to use, and eventually, also in response to demand, a single coherent piece of documentation was written, which later became this manual. The system therefore developed primarily for the programmer to use when working in close association with the lexicographer. Although there are now many things a lexicographer alone can do with the programs, the original orientation is still clearly evident in how one actually uses the programs: for instance in the need to supply "specs" statements in the SNOBOL4 programming language for each run, in the poverty of facilities for detecting and recovering from user-caused and other errors, in the vulnerability of the programs to the user's alteration of internal variables, etc. It also appears in the reliance on specially-written control modules and "user functions" (and in the set of simple basic support functions for writing them) to meet more specialized needs, rather than on a complex and extensive set of pre-programmed facilities. In fact, the arsenal of ready-made programs ("control modules") is extremely limited, compared with what might be needed and could be written. The emphasis instead has been on supplying the underpinnings for the potential additions. In general, the distinction between "user" and "programmer" is blurred. There are clearly things which the user alone can do and things which require a programmer, but between the two poles is a broad grey area in which a varying amount of programming knowledge is required. Also, as was always true in the past, some of the programs in the system--due to more frequent use

and tinkering--are more "finished" than others. No great effort is made to keep the manual in step with the programs as these change and expand. The author insists, however, on staying in close touch with every project that uses these programs, beginning with the initial evaluation of the programs, in order to be apprised of problems and new requirements as well as to ensure that the system will be used in the most effective way. That is another hold-over from the early days of the programs.

1.1.6 ACKNOWLEDGEMENTS

The predecessors of the current system have been developed over many years (since 1966), and many people have made contributions. Early versions of some of the programs were developed by Ann Peters. Almost all of the early users of the programs (see list in Appendix III) contributed important ideas to the system.

Earlier versions of parts of this manual were also written by many people, among them (in chronological order), Byron Bender, Ann Peters, Jim Tharp, June Netter. Joan Romick wrote Appendix II of the mainframe version, and Louise Pagotto wrote Sections 4.4 and 4.5 of the mainframe version. Elaine Good has made substantial improvements to the style and readability of a previous edition of the manual. The excellent presentation of the manual and of the indexes is the handiwork of Louise Pagotto using the Script text formatting package on the IBM/370 and its successors.

Many of the dictionary projects (Appendix III) that occasioned the development of these programs were funded by government agencies, which have thus indirectly contributed to the development of the programs. Among these agencies are the National Science Foundation (NSF), the National Endowment for the Humanities (NEH), and the Government of the Trust Territory of the Pacific Islands. Direct support of the work has been provided by the Pacific and Asian Linguistics Institute and, later, by the Social Science Research Institute, both of the University of Hawaii. The Computing Center of the University of Hawaii has contributed computer time. A major revision and consolidation of the programs and of the documentation were the subject of a grant (BNS 7924700) from the National Science Foundation. Ongoing development incidental to particular projects is often aided by grants to those projects, principally from NSF and NEH.

1.2 ORIENTATION TO THE MANUAL

1.2.1 SCOPE OF THE MANUAL

This Second Edition of the manual comes in two versions, one for the mainframe and the other for the PC environment. Look on the title page to see which version this is. The differences are only in Chapters 4 and 5, and Appendix II.

The principal concern of this manual is to describe the system of lexicographical programs and its use. Ancillary, even though essential, information which is not unique to these programs is only sketchily treated, if at all. (See Section 1.2.2 "What the Manual Does Not Cover".)

The size of the manual in no way reflects the size and complexity of the system it describes. The system itself is quite primitive. The ways in which it can be used however turn out to be not at all obvious to many linguists, and developing this topic and providing some background have greatly expanded the book in many places.

Chapters 2 through 6 of this manual are organized as follows:

Chapter 2, GETTING STARTED, contains a sketch of the minimal information needed to start entering some dictionary entries into the computer and getting some output. This chapter is kept relatively independent of any particular computer, therefore where necessary it refers to Chapter 4 where details of running the programs on a particular computer system are covered.

Chapter 3 is an approximately complete description of the system. Section 3.2 gives all the conventions used in representing dictionary entries, and Sections 3.4 and 3.5 describe what the available programs, known as control modules, do. These two sections are not always entirely up-to-date due to occasional additions to the system. Sections 3.3 and 3.7 attempt to bridge the gap somewhat between this description of the system itself and its application to lexicography.

To actually run a program you need to know the conventions for using the system on a particular computer installation, which may vary from one installation to another. Chapter 4 describes these conventions, in the mainframe version of the manual, for the University of Hawaii Computing Center, and in the PC version, for the MS-DOS set-up. Sections 4.2 and 4.3 show some actual examples of runs called "recipes".

Chapter 5 is a collection of topics oriented to "tasks" that one frequently needs to do but which require some combination of procedures and programs and some skill in deciding among them. These tasks are thus not easily described in any document that is

tied strictly to the structure of the programs. The special "task-oriented" chapter gives the freedom necessary to discuss these matters coherently. Some of the tasks discussed are "generating indexes and finderlists", "storing the data", "sorting the dictionary", etc. This chapter will grow as new tasks are written up.

Chapter 6 gives a technical description of the system. Such a description would be needed by the programmer interested in expanding the capabilities of the system to address new needs and by someone wishing to install the system on a computer.

Finally there are several appendixes and a comprehensive index of topics and parameters. Appendix I should logically be an integral part of Chapter 3, being a complete description of two of the programs, or control modules, INVERT and INVERT2. Due to its inordinate length however, it has been relegated to an appendix. A short summary occupies its place in Chapter 3 instead.

1.2.2 WHAT THE MANUAL DOES NOT COVER

Since certain areas of knowledge or skill, though important or essential to the effective use of the programs, do not relate exclusively to the programs, they are not covered or are only incidentally touched upon in this manual. The principal such areas are the following:

a. Mechanics

This manual does not discuss using a terminal or personal computer, submitting and retrieving jobs and other aspects of using a computing center. However, some information pertaining to the use of one computing center is contained in Appendix II of the mainframe version. Some hints on effective use of a PC are given in the corresponding appendix in the PC version.

The single most important computer tool for anyone using this system of programs, and indeed for almost any computer user, is a program called an editor. Whether on the mainframe or on a PC, it is with the editor that the user prepares and edits the lexical files, composes the short "specifications" file for a run, examines the output from a run, etc. Indeed, with a good and powerful editor, many of the functions of the programs themselves can be performed. One of the skills the user will acquire is that of deciding what functions to perform with the editor and what to entrust to the programs. The decision is mostly a matter of convenience. In any case, a user of these programs should also be quite fluent in using an editor. This manual will not attempt to summarize the capabilities of editors, let alone give details of their use. Nor will we give any advice on the choice of editors, except to state that for someone using these programs, an editor is preferable to a word processor.

b. Job Control Language (JCL) or other operating system command language

This is the language (with differing names on different computers) or other system of commands for instructing the computer to execute programs, to link them together, to look for or deposit data in files, to display or print files, to manage space and other computer resources, etc. To execute any program the user must interact with the computer system through its command language. In this manual, mention of JCL or other command language is restricted principally to Chapter 4. No systematic exposition of this subject is undertaken, however. To understand the command language, to take advantage of its flexibility, and most important, to be able to diagnose abnormal conditions (and in particular to be able to detect whether an abnormal condition originated in the program or in the operating environment), require information beyond the scope of this manual. Appendix II, however, does give some rudimentary help in this regard.

c. SNOBOL4/SPITBOL programming

In almost all of these programs the user may specify certain information by inserting statements written in the SPITBOL programming language. In some cases larger blocks of statements, amounting to small programs in themselves, may or must be inserted. In the former case, the user need only follow by rote the form of the statements as described in Chapters 3 and 4. To take advantage of the latter kind of flexibility, a certain amount of knowledge and skill in SNOBOL4 or SPITBOL programming is necessary, which this manual makes no attempt to impart. The user acquainted with programming, but not with SNOBOL4 will be able to gain the required information by consulting the SNOBOL4 manual (Griswold et al.) and the SPITBOL manual (Dewar) (both are necessary since SPITBOL is the actual language used in these programs but the SPITBOL manual assumes knowledge of SNOBOL4). A growing body of elementary texts in SNOBOL4 is commercially available, but for the complete novice in programming some sort of programming course is recommended.

The basic references are:

Griswold, R.E., J.F.Poage, and I.P.Polonsky. The SNOBOL4 Programming Language. 2nd. ed., Prentice-Hall 1968.

Dewar, Robert B.K. SPITBOL. Version 2.0. Illinois Institute of Technology. 1971.

The following are introductory texts:

Gaskins, Robert, and Laura Gould. SNOBOL4: A Programming Language for the Humanities. Mimeo. University of California Computer Center. Berkeley, 1973.

Griswold, R.E., and Madge Griswold. A SNOBOL4 Primer.

Prentice-Hall 1973.

Hockey, Susan. SNOBOL programming for the Humanities. Oxford University Press 1985

Newsted, Peter R. SNOBOL4: An Introduction to Programming. Rochelle Park 1975.

The following are good for the more advanced programmer:

Gimpel, James F. Algorithms in SNOBOL4. John Wiley and Sons 1976.

Griswold, R.E. String and List Processing in SNOBOL4: Techniques and Applications. Prentice-Hall 1975.

Maurer, W. D. The Programmer's Introduction to SNOBOL. American Elsevier 1976.

Tharp, Alan L. Applications of Spitbol. North Carolina State University 1977.

d. Practical "common sense" about using computers and running programs

This unwritten but essential body of lore may be acquired through experience and by watching or consulting an experienced person in practical situations. The following are examples:

1. When setting up a new run on the mainframe, execute one or more trial runs on small samples of data, both for timing and for testing the instructions in the job, before committing yourself to the expense of a full "production" run. Similarly on a PC, first run a program on a small sample of the data, possibly on a special test file of data, and examine the output to determine that the results are indeed what was desired, before letting the program run on the entire dictionary file. A simple way to limit a run to a small amount of data is to use the STOPAFTER parameter, available for all control modules. Look among the early recipes in Chapter 4 to see how this is used.

2. On inspecting the output from a job, do not assume that the job ran successfully. Check several indicators to make sure that the job ran as desired --- a cursory glance at the output file or printout may not reveal that the run was abnormally or prematurely terminated, or that, even though the job ran successfully, spurious results were given because the original request was formulated erroneously (see Sections 4.4 and 4.5).

3. Keep good records of your mainframe runs and keep the printouts from production runs in some systematic way for future reference. On the PC, follow some systematic method of naming and keeping track of your dictionary files, both input and output files.

Generally practice good housekeeping of your files, both for the sake of managing space and for knowing what's where.

4. Keep track of your tapes and disks; always make back-up copies of your important files and keep them in a different place from the working copies. (This is known as off-site back-up.)

5. In trouble-shooting, or even when not apparently in an abnormal situation, be on guard against taking signs at face-value. Appearances can be deceiving.

6. Be aware that it is always possible that there are still obscure errors in the programs themselves, or in the documentation.

Such precepts are not peculiar to using these programs alone and so are not specially treated in these manuals, though some of these topics may eventually find their way into Chapter 5.

e. Lexicography

This is not a manual on lexicography. It does not give advice on how to compile a dictionary, how to find words, how to write definitions, how to design a practical orthography or decide on an alphabetical order, what information to include in an entry, the role of illustrative sentences, how much cross-referencing to include, how to structure an entry (see, however, Section 3.3), how to get funding or make practical arrangements for a field trip, etc. These are major concerns for the lexicographer but are not brought on (though sometimes they are exacerbated) by the use of a computer. Furthermore, there is almost no field of linguistics or anthropology that does not impinge on lexicography. A bibliography of all relevant materials would be unmanageable. The following is a mixed bag of references on lexicography, showing some of the variety available rather than forming a representative bibliography.

Al-Kasimi, Ali M. Linguistics and Bilingual Dictionaries, Leiden: E.J.Brill, 1977.

Bartholomew, Doris A. and Louise C. Schoenhals, Bilingual Dictionaries for Indigenous Languages, Dallas: Summer Institute of Linguistics, 1983.

Grimes, Joseph E. 'Methods for Semantic Inventories: Huichol'. Technical Memo. Cornell University. 1982(?).

--, Relations and Linkages in the Lexicon. To appear.

Hartmann, R.R.K., ed. Lexicography: Principles and Practices. New York: Academic Press, 1984.

Kiefer, Ferenc, ed., Studies in Syntax and Semantics, Dordrecht: Reidel, 1970.

Landau, Sidney I., Dictionaries: the Art and Craft of

Lexicography. New York: Scribners, 1984.

Mel'chuk, Igor, et al., Dictionnaire explicatif et combinatoire du francais contemporain. Montreal: Presses universitaires de Montreal, 1984.

Misra, B.G., ed., Lexicography in India. Mysore: Central Institute of Indian Languages, 1980.

Saporta, Sol, and Fred W. Householder, eds. Problems in Lexicography, Report on a Conference on Lexicography at Indiana University, 1960. Bloomington 1962 (=IJAL 28, No.2 pt.4). Revised 1967.

Singh, Ram Adhar, An Introduction to Lexicography. Mysore: Central Institute of Indian Languages, 1982.

Sledd, James H., ed. Dictionaries and That Dictionary. Chicago, 1962.

Wierzbicka, Anna, Lexicography and Conceptual Analysis. Ann Arbor: Karoma, 1985.

Zgusta, Ladislav. Manual of Lexicography. The Hague, 1971.

--, ed. Theory and Method in Lexicography: Western and Non-Western Perspectives. Columbia, South Carolina, Hornbeam Press, 1980.

Courses in lexicography are also beginning to be offered at some universities, and a society of lexicography has recently been founded: The Dictionary Society of North America, with its own journal: Dictionaries, the Journal of the Dictionary Society of North America.

Another new lexicography journal is International Journal of Lexicography, published by the Oxford University Press.

The Project on the Lexicon at MIT regularly puts out reports.

f. Other areas of application of these programs

Although these programs were designed specifically for lexicographical work, they need not be limited to the processing of lexicographical data. They can handle other kinds of highly structured lists, such as bibliographies and directories, as well. Since such uses have not been extensively explored and since the main purpose of the programs has so far been lexicographical, this manual contains no systematic treatment of them.

1.2.3 HISTORY AND STATUS OF THE MANUAL

Portions of this manual have been pieced together from documents written at different times by different people, and for slightly different versions of programs. Although the present manual has been edited for accuracy and consistency, some errors have do doubt escaped correction. In any case, not all of the detailed capabilities and limitations of the programs have been included. Additions are often made to the existing programs, and new programs are occasionally incorporated into the system before they are documented in the manual. Conversely, your copy of the programs may antedate the version which your manual describes, so that the manual may mention features not in your programs. Anyone so inclined may examine the programs themselves (the source code) to glean further information from the comments therein.

Beginning with this Second Edition, the manual will appear in two versions, one for the mainframe and one for PC's. Look on the title page to see which version this is. They differ only in Chapters 4 and 5, and Appendix II.

The manual is sporadically being expanded, especially as new programs ("control modules") are made available. Minor revisions to the manual will not warrant a change in the edition number. But copies of the manual printed at different dates may differ in minor ways. The edition number and printing date appear at the bottom of each page.

Comments and questions should be addressed to Robert Hsu, Linguistics Department, University of Hawaii, Honolulu, HI 96822.

1.2.4 WHAT TO READ AND HOW TO START

How to proceed initially depends on the stage of your inquiry into these programs.

If you are trying to determine whether this system of programs would be suitable for your needs without getting involved in the details, there are several things you can do. Read Section 1.1, "Orientation to the System" to get briefly oriented, and Appendix III to get some idea of the lexicographical projects out of which these programs grew. There is a separate "prospectus", available from the author, which attempts to answer the initial questions one is likely to have about these programs. You should definitely talk to previous users of the programs. Some of them have written up their experiences in internal reports or other documents that may be available. The author will be glad to refer you to these users. In fact, I generally insist on discussing with prospective users the suitability of the programs for their needs before turning the programs over to them. Finally, it is possible to get your feet wet

in a useful way without running any programs or even getting near a computer: read Chapter 2, "Getting Started", and try encoding some of your dictionary entries, either on a typewriter or on a computer, in the format required by the programs. Sections 3.2 and 3.3 lead you further into the structuring conventions. You should seek the advice of an experienced user especially during this stage. If you find the conventions useful and congenial, then there is a good chance that you will also find the programs useful.

If you have decided to use these programs, or at least to try them out, they will need to be installed on the computer that you will be using. Write to me for a copy of the programs, which are available for a nominal fee, telling me what computer you wish to run them on. You will also need to purchase a copy of the Spitbol compiler for the particular computer, as well as, for certain computer systems, a sort utility. These details, as well as procedures for installation, can be worked out with me. In the case of the PC version, the details are available in one or two "README"-type files on the distribution diskette. Chapter 6 of this manual also gives some installation information.

If the programs have been installed and you want to make some runs with them, you will need to prepare some data as well as set up a run. Chapter 2, "Getting started", provides some initial guidance in preparing dictionary entries in the required format. Sections 3.2 and 3.3 in Chapter 3 give more extensive information. You may use any editor to prepare the data. (If you use a word processor, make very sure that the file is saved in plain ASCII format, with none of the formatting codes that the word processor normally injects into a document file.) Sections 3.4 and 3.5 describe some programs that are available, while Chapter 4 gives recipes for some common runs. To start with, try setting up and running the three programs suggested at the end of Chapter 2: LISTGEN, BANDSORT, and INVERT2. These tend also to be the most frequently used programs.

CHAPTER 2. GETTING STARTED

This chapter offers a simple way into the system, and refers you to other relevant sections of the manual where particular topics are discussed at greater depth.

Let us assume that you have a collection, possibly preliminary, of dictionary entries written out on notebook pages or on file slips that you wish to enter into the computer for further processing. This has in fact been the most common starting point for users of the system. Assuming the programs themselves have been installed (see section 6.5), there are three classes of things to be done before you can start running programs on this data:

1. purely administrative and mechanical matters, such as, in the case of a mainframe, getting an account, finding a terminal or other data entry device which you can use, learning how to submit jobs to the computer and to retrieve the output, etc.; or in the case of a PC, learning to use the operating system, an editor, and how to manage and print files, etc.

2. decisions about the organization and representation of your dictionary entries; and

3. the actual typing of the entries into computer storage.

We will not address the administrative and mechanical procedures here since they vary from one institution or computer to another and are not specific to these programs, although in Appendix II you will find some instructions and advice.

We will dispose of the third step next, and devote most of this chapter to the second class of decisions since they deserve the most detailed treatment at the start.

It does not matter how the data is entered into the computer. The programs operate on a sequential file of lines (or "records"), the most common form in which computer-readable data is stored. Although in the past such files were entered via key-punch cards, the current method of choice is directly through an interactive "editor" program on the computer. On every computer that can be accessed from a keyboard there is an editor, often several. Describing their use is beyond the scope of this manual.

The second class of decisions, concerning how your dictionary entries are to be represented in a sequential file of lines is, however, central to the use of the programs and will be introduced here.

You first need to identify some of the different types of information that are present in the entries. Common types are

headword (the word being defined), part of speech, definition, synonyms, and illustrative expression. These types of information will be explicitly labelled, in the manner shown below, in the computer representation of your entries. When they are so marked they are known as "bands", in the terminology of these programs. In order to mark them, you must choose a short mnemonic label for each band, e.g., "hw" for headword, "ps" for part of speech, "def" for definition, etc.

You will then be able to start typing entries into the computer in the format illustrated by the following entry:

```
.hw   aba
ps    n
def   a smell or fragrance, pleasant or unpleasant
```

Figure 2.1

The essential features of this "band format" are as follows: 1) Each band begins at the start of a new line (in "column 1", in data processing jargon), beginning with the band name itself. 2) The very first band of a dictionary entry is marked by a period (full stop) in front of the band name. 3) Following the band name, after at least one space, comes the "body" of the band--the information itself.

In the example, the headword of the entry is "aba", the part of speech is "n", and the definition is "a smell or fragrance...".

Additional dictionary entries simply follow on subsequent lines, with the beginning of each new entry indicated only by the initial period. Figure 2.2 shows several entries entered in this format.

```
.hw   aba
ps    n
def   a smell or fragrance, pleasant or unpleasant

.hw   abab
ps    n
def   the leaf of the betel plant
cf    sirih

.hw   abah
ps    n
def   a gap or notch cut in a tree when felling it
```

Figure 2.2

Blank lines may be introduced anywhere, as between entries, to improve the readability, but have no formal significance. Blank lines are simply ignored by the programs.

Note that the second entry in this example has an additional band, labeled "cf", a cross-reference band. Remember that the user may invent band names at will and assign any desired use for the bands. Past users of the programs have assigned bands for such information as pronunciation, etymology, morphological analysis, synonyms, antonyms, conjugation class, usage level (impolite, taboo, etc.), scientific name (of flora and fauna), various types of cross-references, informant name, dialect information, variant forms, unpredictable paradigmatic forms, field notebook reference, specialized meanings (when the word is used as a technical term in some specialized field, e.g. sailing, basketry), semantic domain (kin term, body part, fishing term, etc.), etc. (see Section 3.3.3 for of more possible types of information). Any type of information different from that in existing bands in your file should be accorded its own band. There is no restriction on the number of different bands. Typically dictionaries have grown to use some 40 to 50 different bands, while the largest ones use over 300.

A band name must begin with a letter, and can contain any typeable character except commas, semicolons, and blanks (a blank signals the end of the band name). There is no restriction on the length of band names. In general, short band names may be faster to type but harder to remember.

The choice of band names is not irrevocable. You may change the names of existing bands or add new bands whenever you wish. You are in fact encouraged to add a band whenever you discover a new type of information, and to revise your system of bands for greater clarity and consistency. Furthermore, you are encouraged to discover new types of information in your file, even though this may mean going back and splitting up instances of an existing band. Note that a detailed categorization of information is not dictated by the computer, nor is it a prerequisite for using these programs. However, the finer the categorization and structuring the more enlightening and useful the file will be, both in itself and as an object to explore.

Not every dictionary entry of course need have the full complement of bands. If, for instance, an entry does not (yet) have a definition, then the corresponding band can simply be left out of that entry. Some lexicographers proceed by making a skeleton wordlist first, containing only a headword band and a short definition band in each entry. Working with an informant, they then make several passes through the printout, eliciting information for a particular band in each pass.

In the example in Figure 2.2, the body of each band was quite short. What if the body of a band is so long that it cannot fit on one physical line of whatever display or storage device is being used? It must then be continued on the next line. However, if the line continues in column 1 of the next line, the programs will mistake it for the name of a band. (See Figure 2.3.)

```
def    the wild vine, more commonly, the leaf of
the betel plant
```

Figure 2.3

In this example, there would be no way to tell that "the" on the second line is not the name of a band. Instead, the convention for a continuation line is to begin after two spaces (i.e., in "column 3"): the first space signals a continuation line, and the second space is the normal space between the last word of the previous line and the first word of the continuation. There may be as many continuation lines as needed for a given band. This method of continuation is shown in Figure 2.4.

```
def   the wild vine, more commmonly, the leaf of
  the betel plant used to wrap areca nut shavings
  and lime and chewed
```

Figure 2.4

Do not hyphenate a word at the end of a line; the hyphen would become part of the word.

This representation of a band as one or more lines is called a "hanging paragraph," which is more completely described in Section 3.2.

A paragraph may, of course, be no more than one line long: it then does not have any continuation lines. Every band is a separate paragraph. Conversely, every paragraph in a dictionary file represents a band.

One consequence of this method of representation is that the point where a paragraph is broken across lines is not significant; lines may be of any desired length. The above band may also be represented as follows:

```
def   the wild vine, more commonly, the
  leaf of the betel plant used
  to wrap areca nut shavings and
  lime and chewed
```

Figure 2.5

As far as the programs are concerned, Figures 2.4 and 2.5 above represent the identical information.

It is possible to represent entries that are more complicated than a simple list of bands. Section 3.2 describes the conventions for grouping bands within an entry and for making subentries. We

will, however, not take up grouping and subentries in this introductory chapter.

A dictionary file, then, is simply a sequential file of lines such as those in the above examples. It is entered in this form and edited in this form, i.e., as nothing more than a sequence of lines. The programs will, however, understand the structure. They group lines together into a paragraph, recognize anything that begins in column 1 as a band label, look for the period to mark the start of a new entry, etc. Before we discuss a few programs that can be run on such a file, we will bring up some additional decisions you may want to make before entering the bulk of your material.

If the material in certain bands uses letters and other characters not available on your terminal or other input device, you may need to establish some conventions of orthographic equivalence for those bands. For instance, a common practice is to indicate stress by putting an apostrophe (also called a single quote) after the vowel affected. Another is to use "?" for the glottal stop. Syllable breaks may be shown by a space. The following example of a pronunciation band shows these symbols in use:

```
|pr   a ba'?                                                    |
```

Figure 2.6

Note that the apostrophe need not mean stress in all bands, but only in those bands that you choose. Elsewhere it may be used to mean other things. In general, you may establish your own conventions, orthographic and otherwise, on a band by band basis. This is known as the 'band-by-band principle' (see Section 3.3.8). Section 5.2 will contain some suggestions about how to handle orthographic problems.

For all but the simplest bands you may want to establish additional conventions for punctuation, abbreviations, order of material within the band, etc. Again, the conventions may be made on a band by band basis. You do not need to and probably will not be able to make all these decisions before you start. You may postpone decisions that are not yet relevant. However, it will save much confusion and editing later if every convention is written down and easily accessible, say in a codebook, and if you make sure that everyone involved with entering data and proofreading understands and follows them. A codebook might be a looseleaf notebook with a separate page (or pages if necessary) for each band, arranged alphabetically by band name. The band name would appear prominently at the top of a page, followed below by a short description of the type of information that it represents. Elsewhere on the page would be given all the conventions that apply within that band. Any conventions external to the band would also be noted, such as whether the band is obligatory, whether a certain other band must follow, etc. If certain conventions or sets of conventions occur

frequently, they may be given names, defined in a separate section of the codebook, and referred to only by name on the band-convention pages.

Certain conventions should be established early on. An example of such a decision occurs when the lexicographer wishes to anticipate the eventual setting into type of the dictionary. It will be a simple matter, when that time comes, to instruct the computer to set the body of certain bands in boldface, and other bands in italics, etc. However, if, say the first part of a band is to be set in italics and the rest in another font, then the point of change must be explicitly marked in the body of the band. This marking is more efficiently done when the material is first entered. In a typical illustration band, for instance, the illustration itself is often set in italics while the translation which follows it is in roman, also called medium. While the italic shift does not need to be marked because that will always occur at the beginning of the band, the point of shifting to medium (somewhere in the middle of the band) does have to be marked. This is often done with the vertical bar symbol, "|", as in the "il" (for illustration) band in Figure 2.7.

```
|.hw  aba'b                                                    |
|df   the wild vine, more commonly, the leaf of the           |
|  betel plant                                                 |
|il   abab dia agih  |two bunches of betel leaves              |
```

Figure 2.7

Similarly, a shift to italics, as frequently encountered with Latin binomials, is usually indicated by the percent sign, "%", as shown in Figure 2.8.

```
|df   the wild vine, more commonly, the leaf of               |
|  the betel plant (Sp. %Piper betle|) used to wrap           |
|  areca nut shavings and lime and chewed                      |
```

Figure 2.8

Note the shift back to medium before the closing parenthesis in this example.

Although most users of these programs use "|" and "%" to indicate shifting to medium and to italics for the anticipated typesetting of their work, the convention is not dictated by this system of programs; you may use any other symbols for this purpose if, for instance, you need "|" or "%" to represent other information in the band. Other conventions are discussed in Section 5.2.

There is another commonly used convention, one which is more firmly entrenched. It is the use of the asterisk, "*", to mark, in

definition bands, words that are to be extracted by the "finderlist" program. This program constructs an alphabetical index of such words, called keywords, which indexes back to the headwords whose entries contain those words. The definition band in Figure 2.9 illustrates the use of the asterisk for marking keywords:

```
.hw   abab
def   the wild *vine, more commonly, the
  *leaf of the *betel plant used
  to wrap areca nut shavings and
  lime and chewed
```

Figure 2.9

The finderlist generated from a dictionary containing this entry would have entries for "betel", "leaf", and "vine", each containing a reference to the headword "abab". A complete description of these and other conventions assumed by the finderlist program, including possible overriding of the conventions, is contained in Appendix I.

Note that asterisks are used for this purpose only in definition bands. In other bands asterisks may be used for other purposes, such as to mark reconstructed forms. This is another example of the band-by-band principle, whereby a convention need not be applicable to all bands in a dictionary but only to certain stated bands.

Again, it should be emphasized how important it is to write down all band-specific conventions in a codebook.

Using the few conventions discussed above, you might now have some entries that look like this when entered into the computer:

```
|.hw   aba
|pr   a ba'
|df   a *smell or *fragrance, pleasant or unpleasant
|cf   bau
|
|.hw   aba'
|pr   a ba'?
|df   *at, *atop, *in, *on
|phr   aba' mija, |on the table
|phr   aba' geladak, |on the floor or deck
|phr   aba' jelatong, |on the jetty
|
|.hw   abab
|pr   a ba'b
|df   the wild *vine, more commonly, the leaf of the
|  *betel plant (Sp. %Piper betle|) used to wrap
|  areca nut shavings and lime and chewed
|cf   sirih
|ncl   agih
|phr   abab dua agih, |two bunches of betel leaves
|
|.hw   abah
|pr   a ba'h
|df   a gap or *notch cut in a tree when felling it
|il   Abah ia dalam, |his notches are very deep.
```

Figure 2.10

The "ncl" band name in Figure 2.10 stands for 'numeral classifier'.

If you are unfamiliar with the terminal or other data entry device you will be using, you should first type up entries on paper in this format before sitting down at the terminal.

Once a file like this is in computer storage, what are some useful programs that can be run on it? In this chapter we will mention the three most commonly used programs.

The first is called LISTGEN, which prints the file out on paper (or prepares another file ready for printing or examination on the screen) with some indentations and blank lines to show the structure of each entry more vividly than does the input format. The Listgen format (l-format) is the usual form for working copy--the printout on which editing is done. Making allowance for the artificially narrow space in the example box, the above long sample in Figure 2.10 would come out looking something like this (Figure 2.11) in l-format:

```
1  .hw   aba
          pr    a ba'
          df    a *smell or *fragrance, pleasant or
            unpleasant
          cf    bau

2  .hw   aba'
          pr    a ba'?
          df    *at, *atop, *in, *on
          phr   aba' mija, |on the table
          phr   aba' geladak, |on the floor or deck
          phr   aba' jelatong, |on the jetty

3  .hw   abab
          pr    a ba'b
          df    the wild *vine, more commonly, the leaf of
            the *betel plant (Sp. %Piper betle|) used to
            wrap areca nut shavings and lime and chewed
          cf    sirih
          ncl   agih
          phr   abab dua agih, |two bunches of betel
            leaves

4  .hw   abah
          pr    a ba'h
          df    a gap or *notch cut in a tree when felling
            it
          il    Abah ia dalam, |his notches are very deep.
```

Figure 2.11

Note that there is neither more nor less information in this, nor is any information rearranged. It is merely in a more readable format. Also note that the LISTGEN program numbers the dictionary entries. The listgen output is of course a separate file; the original file is left untouched, and continues to be the master file for editing. None of the programs alter the master file.

How exactly you run the LISTGEN program depends on your computing system and on how the programs were set up to run there. Instructions are given in Chapter 4. In particular, see the "recipes" section, 4.2, for the simplest LISTGEN run.

Another commonly used program is called INVERT (or the similar but more advanced INVERT2), which generates an alphabetized "finderlist" using the asterisked words in the definition bands. If run on the sample dictionary in Figure 2.10, INVERT would generate an output that looks something like Figure 2.12. Note that every starred word in a definition band appears in this alphabetical list of English words, and that the entire definition appears as many times as there were stars in it. The stars are dropped in the finderlist, however.

```
at
          at, atop, in, on::  aba'

atop
          at, atop, in, on::  aba'

betel
          the wild vine, more commonly, the leaf of the
            betel plant (Sp. %Piper betle|) used to wrap
            areca nut shavings and lime and chewed::  abab

fragrance
          a smell or fragrance, pleasant or unpleasant::  aba

in
          at, atop, in, on::  aba'

notch
          a gap or notch cut in a tree when felling it::
             abah

on
          at, atop, in, on::  aba'

smell
          a smell or fragrance, pleasant or unpleasant::  aba

vine
          the wild vine, more commonly, the leaf of the
            betel plant (Sp. %Piper betle|) used to wrap
            areca nut shavings and lime and chewed::  abab
```

Figure 2.12

A larger dictionary would have produced a more interesting finderlist having typically many more entries under each keyword. To run this program you need to tell the program what bands to look for asterisks in, in this case only the band "df". To do this, you need to place a statement into the program that looks like this:

```
BANDS  =  'DF'
```

See Section 4.3 for a recipe for running INVERT. Appendix I gives instructions on specifying the bands in which the program is to look for asterisks.

There are other symbols besides asterisks you can put in your definition bands to guide the program in picking out keywords and the longer phrases in which they are embedded. These are described in Appendix I, which is the complete document on using the INVERT and INVERT2 programs.

The final program to be mentioned in this introduction is called BANDSORT. It simply sorts all the bands in a dictionary file as if they were separate items and not part of any entry. It does, however, carry along the headword, placing it at the left of each band, so that you can tell which entry each band came from. The bandsort from our sample file would look something like Figure 2.13:

```
|aba        cf    bau
|abab       cf    sirih
|
|aba'       df    *at, *atop, *in, *on
|aba        df    a *smell or *fragrance, pleasant or unpleasant
|abah       df    a gap or *notch cut in a tree when felling it
|abab       df    the wild *vine, more commonly, the leaf of the
|                   *betel plant (Sp. %Piper betle|) used to
|                   wrap areca nut shavings and lime and chewed
|
|aba       .hw    aba
|aba'      .hw    aba'
|abab      .hw    abab
|abah      .hw    abah
|
|abah       il    Abah ia dalam, |his notches are very deep.
|
|abab       ncl    agih
|
|aba'       phr    aba' geladak, |on the floor or deck
|aba'       phr    aba' jelatong, |on the jetty
|aba'       phr    aba' mija, |on the table
|abab       phr    abab dua agih, |two bunches of betel leaves
|
|aba        pr    a ba'
|aba'       pr    a ba'?
|abab       pr    a ba'b
|abah       pr    a ba'h
```

Figure 2.13

In this output, all occurrences of the same band are grouped together and subsorted on the body of the band itself. The headword of the entry to which the band belongs appears at the far left. Many lexicographers have felt that this is one of the most widely useful printouts. It can be used to find all entries having the same or similar information in a given band, e.g. all entries with "n" in the "ps" band. It can also be informative to look at those entries with bands which occur only rarely. The output of a bandsort is also useful as a proofreading aid, allowing one to scan only one type of band at a time paying attention to only the conventions relevant to that band.

There is a recipe for running the BANDSORT program in Section 4.3.

This chapter has deliberately been kept short. At this point you may find it instructive to encode a few dictionary entries of your own, possibly trying more than one way to translate your entries into the band format. As you encounter practical problems in this process, you may find the more detailed discussion in Sections 3.2-3.3 helpful. You should also seek the advice of someone experienced with the system before you spend too much time entering your materials, in order to catch potential problems and misunderstandings.

After you have worked up one or two dozen representative entries in this way, type them into a computer file observing the conventions described above and in the next chapter. Then run a LISTGEN, a BANDSORT, and an INVERT on the file, and examine the output, again preferably seeking the advice of an experienced user. This may result in modifications to your encoding scheme. Make the changes, and add a few dozen more entries, and run the programs again.

By now you should have started your codebook.

Even after your conventions seem to have stabilized, continue to run these programs periodically on your growing database and scan the outputs. You may discover patterns in your material which you would want either to take advantage of in some way or to alter. Do not wait until all your material is in before running a set of programs. The usefulness of the programs lies in bringing out patterns while the work is in progress, and not so much in producing a "final" product. The three programs mentioned above are probably the most useful for any dictionary. For an individual language other programs may also prove to be heuristically useful.

CHAPTER 3. THE USER'S VIEW OF THE SYSTEM

This chapter gives a detailed description of the system from the user's point of view. (Chapter 6 gives a technical view for the programmer.) The first section, 3.1, describes the overall organization of the programs. Sections 3.2 and 3.3 set out the conventions for representing lexical data in a computer file, while sections 3.4 through 3.7 have to do with the actual programs and their use.

3.1 STRUCTURE OF THE SYSTEM

The system consists of two general programs, BIRDBATH and TOAST. BIRDBATH operates by reading through a dictionary file from beginning to end, performing specified operations on the data read in. (The file itself is never altered.) These operations may involve formatting the data for printing, selecting entries on the basis of certain criteria, copying certain types of information into another file, etc. This program does not do any sorting, that is, rearranging of the data. To sort or to do anything requiring sorting the other program is used. That program, TOAST, starts by also reading through a dictionary file, but in the process it generates a file suitable for sorting. It then sorts this file and finally processes the re-ordered file. (Again, the original file is left unchanged.)

Control Modules

When you run BIRDBATH, you must tell it what type of operation (such as the three mentioned above) you want it to do. This is done by telling it to use a certain "control module" corresponding to the desired type of operation. For instance, LISTGEN is the name of a control module that formats the dictionary in the standard printout format. The control module is actually a small piece of program that is inserted into the main BIRDBATH program and that controls what operation BIRDBATH does on a dictionary file. Similarly a TOAST control module is a piece of program inserted into the TOAST program.

There is a collection, or "library", of control modules, each of which does some particular type of thing when inserted into BIRDBATH. There is a separate library of control modules for TOAST. Whenever a new need is encountered that is not addressed by an existing module, either an existing module can be generalized to satisfy that need or a new module must be written. BIRDBATH and TOAST were designed with the goal of making such modules easy for a programmer to write. If the new module is a specialized one which is likely never to be needed again, it can be simply thrown away after it is used, but if it is of general interest it may be incorporated into the standard library of modules and a description of it would be added to this manual. You may also accumulate your

own collection of modules. In Sections 3.4 and 3.5 you will find a list of existing standard control modules for BIRDBATH and TOAST, respectively, with a description of what they do and the options available with each.

If you need to write your own control module, you have to know a certain amount of SNOBOL4 programming and how BIRDBATH and TOAST work. The latter is explained in Chapter 6.

SPECS

When using a control module, in addition to giving the module's name you may need to give more specific information, the so-called SPECS parameters, to specify in greater detail what the program should do. For instance, for a control module that selects only certain bands of a dictionary you need to specify the names of these bands. The SPECS parameters relevant to each control module are discussed with each module (see Sections 3.4, 3.5, 3.6).

The SPECS statements are regular statements in SNOBOL4 or SPITBOL, the programming language in which the entire program system is written. These statements become part of the program. When you write such statements you should use the example SPECS statements (shown in various places in Chapters 3 and 4) as models, especially with regard to the placement of spaces (blanks). Where spaces are shown, be sure to leave at least one space (more than one if you wish, but at least one); where you see no spaces, leave none. This also applies to the beginning of the statement. Most statements, as shown, begin with spaces (do not begin in column 1 or the very left of the line). Those that do start in column 1 are shown starting at the left margin in this manual.

USERFNS

Certain kinds of specific information of a more complex kind, typically requiring the description of procedures rather than the specification of values, may be required by some control modules. Such information is called "user functions" and these are usually placed in a file designated USERFNS.

Job control or command language

Finally, there is more general information not specific to BIRDBATH or TOAST that you have to provide via Job Control Language (JCL for short) or other command language understood by your computer system, in order to tell it that you want to run BIRDBATH or TOAST, what files to use, and other administrative details.

This language and associated rituals vary from computer to computer, depending on how the programs were installed, idiosyncracies of the computer system, etc. General instructions therefore cannot be given here. Chapter 4 and Appendix II provide some guidance for specific systems. Chapter 6 contains some notes that may also be helpful.

Before we describe the available control modules we need to describe the form of the data that the programs operate on, i.e. how a dictionary is represented in a computer file. Sections 3.2 and 3.3 will be devoted to that.

3.2 ORGANIZATION AND REPRESENTATION OF DICTIONARY ENTRIES

3.2-1 BAND FORMAT CONVENTIONS

Lexical information is represented in a computer dictionary file with the conventions described in this section. Each entry in the dictionary consists of a sequence of "bands". A band usually corresponds to one of the categories of information present in the dictionary, such as part of speech, definition, synonyms, etymology, cross-reference, and even the headword itself. There is potentially an unlimited number of categories one might be able to identify. It is up to the user to decide what categories to label in a given lexical file.

a. Band names

For each band the user invents a short mnemonic name, typically of two to four letters. The band name must begin with a letter and can contain any typeable characters except blanks, commas, and semicolons. Do not use upper and lower case to distinguish band names. Capital and lower case band names are normally not distinguished from each other by the programs. For instance, DF, Df, dF, and df are all considered to be the same band name by most of the programs. For clarity in text we tend to cite band names in caps even though they may be in lower case in the Figures. (Of course, capitals and lower case letters are considered distinct in the rest of the band, the "body" of the band.) The following might be appropriate band names for the categories of information mentioned in the last paragraph: PS, DF, SYN, ETY, XR, HW.

Three band names have special meanings in certain contexts, and it would be simplest to avoid them except for those purposes. They are TITLE, FILE and LIST. Their uses are described in Section 3.3.3, Uses of Bands.

b. Body of bands

Each band consists of a band name followed, after one or more spaces, by the "body" of the band, which may contain any typeable material. The body of the band represents the actual information (the part of speech, etymology, etc.) in the band. The user may wish to establish a particular format and other conventions to be observed in each band. Some conventions have become customary, and certain others are expected by some of the programs. The conventions are discussed in the sections where they are applicable. Thus, a band may look something like this:

```
df   silence, stillness
```

Figure 3.1

In this example, the band name is DF, and the body of the band consists of "silence, stillness".

There must be at least one blank separating the band name from the body. For clarity we usually leave about three, and the programs, when producing band-format output, insert a variable number in an attempt to line up the start of the body at a fixed column. The actual number of blanks is not significant. In fact, it is not possible for the first character of the body to be a blank. Nor for the last.

The body of a band may be left empty. This might be done, for instance, if the pertinent information is not yet known for a particular entry and it is desired to explicitly signal this fact. Of course the entire band, name and all, may be left out. There is no way, however, to leave off the band name itself and enter only the body of the band.

c- Entries

An entry in the dictionary is a sequence of bands that belong together, typically because they all pertain to the same headword. Normally the first band of the sequence would contain the headword. The first band of an entry is formally distinguished by having a period (full stop) before the band name--with no intervening space. The following group of bands would constitute an entry:

```
.hw   fi:le:mu:
ps    NOM
df    silence, stillness
il    ko te fi:le:mu: o te po |the silence of night
```

Figure 3.2

The headword is "fi:le:mu:". (It is a common convention to use ":" to indicate vowel length.) The part of speech is "NOM", and the definition is "silence, stillness". Another common convention, one that maximizes the amount of information coded, is to not capitalize the first word of a definition unless the word is a proper name or otherwise normally capitalized. There is an illustration (IL) band with a short phrase and its translation. (Another common convention: in illustration bands the beginning of the translation is marked by a "|" sign.)

The headword band need not, of course, be named HW. In fact, even within the same dictionary the headword band can go by different names if desired, for instance, to indicate what kind of lexical unit it is: RT for root, SUF for suffix, CPD for compound, etc.

Not all the bands used in a dictionary need be represented in each entry. Of the dozens of different bands used, often only five

or six will appear in any given entry. The bands should appear in a consistent order. However, ordering of bands is a matter of preference, and is not enforced by any of the programs. There is a control module that can be used to put them into a specified order. A given band may be repeated within the same entry if multiple instances of the same category of information are to be included. A typical case is when there are several illustrations: then the IL band (if that is the name of it) can occur more than once, with a different body each time.

Entries simply follow one after the other in the file without a break, the period before the first band of each entry being the only thing to mark the start of a new entry.

d. Hanging paragraphs

Each band is represented in a computer file as a "hanging paragraph". Roughly, this means that it begins at the very left of a line, i.e., without any initial blanks, and if the band needs to be continued onto subsequent lines those lines must normally start with two blanks. The first of these blanks signals a continuation line, and the second is the blank between the last word on the previous line and the first one on the continuation line. If you want to have two blanks between the last non-blank character on one line and the first non-blank character on the continuation line, then begin the continuation line with three blanks. The first blank is, again, the continuation indicator, and the remaining two are "real" blanks. Within the paragraph then, i.e. apart from line-end blanks (which are ignored) and the single initial blank of continuation lines, blanks are taken as significant data--part of the contents of the paragraph, and one blank is different from two blanks, and so forth. It is irrelevant where the paragraph is broken across lines however. Thus, there is freedom to break paragraphs according to the available width of the paper, screen, storage device, etc., on which the paragraph is printed, displayed, or stored. In Chapter 2 we displayed two representations of the identical paragraph. See Figures 2.4 and 2.5.

The complete set of hanging paragraph conventions are given in Section 3.2.3.

e. Modes

It is often necessary to distinguish separate groups of bands within an entry. If, for instance, there is a definition and an illustration for the noun use of a word, and a similar set of bands for the verbal use, one would want to show that the first set of 3 bands (part of speech, definition, and illustration) belong together, and similarly for the second set. This is done by putting numerals in front of the band names. All bands having the same numeral before the name belong together. Please note in Figure 3.3 how the numerals serve to make three groups of bands.

```
|.hw   fi:le:mu:
|1ps   NOM
|1df   silence, stillness
|1il   ko te fi:le:mu: o te po |the silence of the
|  night
|2ps   VB
|2df   be quiet
|2il   e fi:le:mu: te vahega |the class is quiet
|3ps   QUAL
|3il   he po: fi:le:mu: |a silent night
|3il   havali fi:le:mu: |walk silently (or quietly)
|3il   fenua fi:le:mu: |peaceful land
```

Figure 3.3

Each group of bands with the same number before it belongs to the same "mode", and the numerals are called mode prefixes. There are no formal restrictions on the use of mode prefixes. Any constraints are more a matter of whether you can put a reasonable interpretation on a given use. It should also be pointed out that the interpretation put on the modes (groupings of bands) is not prescribed; you may use modes (and submodes, see below) for whatever purpose or meaning you wish. Formally, they are simply groups of bands, no matter for what reason one may want to group them. A mode is strictly a contiguous group of bands; one cannot have mode 1 bands, for instance, separated by bands of mode 2.

It makes most sense to number all modes beginning with 1 and increasing. However, frequently bands may be encountered at the beginning of an entry that do not have any mode numbers; this implies that these bands "belong" to the whole entry and not to just one group of bands within it. Typically these might be pronunciation bands, or etymology, or alternate forms. There may also be mode-less bands at the end of an entry; these might be cross-reference or synonym bands.

If the mode numbers become greater than 9, one may start from 1 again. Do not go to double digits, i.e., 10, 11, etc., since two digits indicate submodes, and three digits sub-submodes, etc. (see below). It is rare to need more than 9 modes, however.

f. Submodes

A more common need is for sub-modes (and sub-submodes, etc.), i.e. for groups within groups. A second (and third etc.) digit is used to gather groups of bands within a primary (secondary etc.) group. Figure 3.4 shows submodes. Study the numbering system carefully.

```
|.hw   fi:le:mu:
|1ps   NOM
|11df   silence, stillness
|11il   ko te fi:le:mu: o te po |the silence of the
|  night
|12df   peace
|12il   ko te fi:le:mu: o te Atua |the peace of God
|2ps   VB
|21df   be quiet
|21il   e fi:le:mu: te vahega |the class is quiet
|22df   be in peace
|22il   e fi:le:mu: te lalolagi
|23df   be gentle
|23il   nae fi:le:mu: lele te puhi |the cat was very
|  gentle
|3ps   QUAL
|3il   he po: fi:le:mu: |a silent night
|3il   havali fi:le:mu: |walk silently (or quietly)
|3il   fenua fi:le:mu: |peaceful land
```

Figure 3.4

g. Formats

Although this format, known as p-format (paragraph format), is rather difficult to read, it is easy to type in and to edit, and is simple for storage and programming purposes.

However, one does not normally work from a printout such as the above. One of the programs, called LISTGEN, produces a formatted printout which is much easier to work with, containing indentations and blank lines, and plenty of white space for writing in. The example entry in Figure 3.4 would be formatted by LISTGEN as shown in Figure 3.5. Please compare the two figures. This format is known as LISTGEN format or l-format.

```
.hw   fi:le:mu:
            1ps   NOM
              11df   silence, stillness
              11il   ko te fi:le:mu: o te po |the
                silence of the night

              12df   peace
              12il   ko te fi:le:mu: o te Atua |the
                peace of God

            2ps   VB
              21df   be quiet
              21il   e fi:le:mu: te vahega |the
                class is quiet

              22df   be in peace
              22il   e fi:le:mu: te lalolagi

              23df   be gentle
              23il   nae fi:le:mu: lele te puhi |the|
                cat was very gentle

            3ps   QUAL
              3il   he po: fi:le:mu: |a silent night|
              3il   havali fi:le:mu: |walk silently |
                (or quietly)
              3il   fenua fi:le:mu: |peaceful land  |
```

Figure 3.5

h. Sub-entries

Another formal device available is that for indicating sub-entries. A sub-entry is typically used for a derived word (a word derived from the main entry headword). A sub-subentry would be for a word derived from that derived word, etc. A sub-entry looks exactly like a main entry (the type we have been assuming) and can have all of the band and mode structure, except that its first band begins with two periods (full stops) instead of one. A sub-subentry begins with three periods, etc. There can be as many sub-entries as desired. They are placed consecutively after the main entry to which they are subordinate. Sub-subentries subordinate to a given subentry are placed after that subentry. And so on. This can go on to any depth--by adding more periods. When printed out by the LISTGEN program, the entire sub-entry is indented a little further than the main entry, and sub-subentries are indented a little more, etc. Figure 3.6 shows a main entry followed by three subentries, in the input format. Figure 3.7 shows a LISTGEN of the same material. Please compare them.

```
|.hw   ahu
|1ps   n.
|1df   heap, mound
|2df   sacred mound used in certain rites
|2il   Tohinga mauritanga o papa ma ki mua ki te
|  ahu  (M. 248).
|3ps   v.t.
|3df   heap up
|3il   Katahi ka ahu raua i ta raua tuaahu
|  (W. v, 75).
|4ps   a.
|4df   heaped up
|4il   He mea ahu nga onepu e nga ringaringa o te
|  tohunga (M lxxxiii).
|..hw  ahunga
|ps    n.
|df    heaping up
|il    te ahunga oneone, te aponga ki punga ra
|  (M. 127).
|..hw  whakaahu
|1ps   v.t.
|1df   heap up, lay in a heap
|1il   Whakaahua koe ki te ahi rarauhe (M. 5).
|1il   Whakaahutia he aruhe ki runga ahi.
|2ps   v.i.
|2df   swell up
|3df   be displeased, dissatisfied
|3il   Katahi te tangata whakaahu ki aku kai.--Whaka
| ahu ana au i te pukapuka a Henare: ko taku whaka
| ahu tenei, te haere ia ki Wharekahika kati mai ai.
|..hw  ahuahu
|1ps   v.t.
|11df  heap up
|11il  He mea ahuahu nga puke hei taunga mo a ratou
|  teka.
|12df  earth-up crops, etc.
|2ps   n.
|2df   the process of earthing up
|2il   I u mai a Paikea ki enei motu ... i te wa i
|  tupuketia ai te ahuahu (W. iii, 35).
```

Figure 3.6

```
.hw   ahu
            1ps   n.
            1df   heap, mound

            2df   sacred mound used in certain rites
            2il   Tohinga mauritanga o papa ma ki mua ki
              te ahu  (M. 248).

            3ps   v.t.
            3df   heap up
            3il   Katahi ka ahu raua i ta raua tuaahu
              (W. v, 75).

            4ps   a.
            4df   heaped up
            4il   He mea ahu nga onepu e nga ringaringa
              o te tohunga (M lxxxiii).

  ..hw  ahunga
             ps   n.
             df   heaping up
             il   te ahunga oneone, te aponga ki punga ra
               (M. 127).

  ..hw  whakaahu
               1ps  v.t.
               1df  heap up, lay in a heap
               1il  Whakaahua koe ki te ahi rarauhe (M. 5).
               1il  Whakaahutia he aruhe ki runga ahi.

               2ps  v.i.
               2df  swell up

               3df  be displeased, dissatisfied
               3il  Katahi te tangata whakaahu ki aku kai.
                --Whakaahu ana au i te pukapuka a Henare:
                 ko taku whakaahu tenei, te haere ia ki
                 Wharekahika kati mai ai.

  ..hw  ahuahu
               1ps  v.t.
                 11df  heap up
                 11il  He mea ahuahu nga puke hei taunga mo
                   a ratou teka.

                 12df  earth-up crops, etc.

               2ps   n.
               2df   the process of earthing up
               2il   I u mai a Paikea ki enei motu ... i te
                 wa i tupuketia ai te ahuahu (W. iii, 35).
```

Figure 3.7

3.2.2 BAND FORMAT CONVENTIONS: SUMMARY

Below is a brief summary of the conventions for representing entries.

1. Each type of information (headword, part of speech, definition, etc.) starts on a separate line. This representation of a type of information is called a BAND.

2. Each type of information is labeled with a short BAND NAME of your choice. The name must begin with a letter, and may contain any characters except blanks, commas, and semicolons. Upper and lower case have no distinguishing value in band names.

3. Not every band used in a dictionary need be present in every entry.

4. A band may be empty -- band name only, no body.

5. A band may be repeated within an entry.

6. The information within a band is called the BODY of the band. You may subdivide the body of a band into fields if desired, according to any convention you choose.

7. The order of bands within entries should be, though need not be, consistent from entry to entry.

8. Bands within an entry may be grouped and subgrouped to any depth. A set of consecutive bands belonging to the same group is indicated by having the same numerical MODE PREFIXES before the band names (no intervening space). Top level, no prefix. First level 1,2,3, etc.; second level 11,12,13,...,21,22,23, etc.; third level 111,112,113,...,121,122,123,...,211,212,213, etc.

9. The first band in an entry, usually the headword band, is marked by a period before the band name, e.g., .HW

10. The first band of a sub-entry is marked by two periods, i.e., ..HW, sub-subentries are marked by three periods, etc.

11. The hierarchy of subentries is indicated only by physical order: e.g., a sub-subentry is considered subordinate to the closest previous subentry. Subentries always follow the main entry to which they are subordinate.

12. The prefix (if any), the periods (if any), plus the band name (no blanks in any of this) constitute the BAND LABEL. The band label is separated from the body of the band by one or more blanks. The exact number used is not significant, as long as there is at least one.

13. A dictionary file consists only of a sequence of bands. It is divided logically into "entries" only by the fact that the first band of each entry has a single period preceding the band name.

14. There may be any number of entries in a dictionary file; there may be any number of bands in each entry; and there may be any number of different bands used in the dictionary.

15. Each band, starting with the band label, is represented on records (i.e. lines) in a file as a single "hanging paragraph" as described below.

3.2.3 HANGING PARAGRAPH CONVENTIONS

The following conventions are used for representing variable-length units of text, such as bands, on limited-length records, e.g., punched cards, lines in disk files, or printed lines on paper.

1. The paragraph starts in column 1. It may not start with a blank.

2. If a paragraph exceeds one record, continue in column 2 of the next record (leaving column 1 blank). Any blanks at the end of a record are ignored ("non-significant blanks"). Column 2 of the continuation record is considered to come right after the last non-blank character on the preceding record.

3. As many continuation lines may be used as needed to accommodate a paragraph.

4. All blanks within a paragraph, i.e., all blanks other than line-end blanks and the single initial blank of continuation lines, are treated as significant blanks.

5. A fixed-length field may be reserved at the beginning of each record for identification or other information. It will not be processed by the programs. "Column 1" would then refer to the first column to the right of this field.

3.3 GUIDE TO USING BAND FORMAT

This section explores in greater detail the ways in which the format conventions described above may be used in encoding dictionary entries. We point out alternative ways of representing certain structures, as well as some limitations of the formalism. We also mention some lexicographical problems that encoding of a dictionary naturally brings up, though since they are linguistic rather than data structuring problems, no attempt is made to offer solutions.

3.3.1 POLICIES ABOUT ENTRIES AND HEADWORDS IN THE DICTIONARY

These are primarily lexicographic matters, but the fact that the dictionary will be computer-processed could have something to do with how you decide to resolve them.

(Note that by "entry" we mean the entire body of material associated with a given headword. By contrast, in some of the lexicographical literature, and in the front matter of some dictionaries, "article" is used in place of our "entry", while "entry" or "entry word" refers to only the headword.)

The first question concerns the kind(s) of lexical unit: What units is this a dictionary of? That is, are the main headwords to be roots, stems, words, or other kinds of units, or all of these? What kinds of derivatives and compounds will be included as headwords, with their own main entries or subentries? Will (some) idiomatic or frozen expressions get their own main entries, or will some of them be relegated to subentries, or even to bands within an entry? In the case of languages with highly productive word derivation, and of polysynthetic languages, which of the many long morphological constructions will be chosen as headwords?

Closely related is the question of how to treat derivatives and compounds, and, in general, "families" of related items. Are derivatives--i.e., words derived from roots or other words--to be given their own main entries or will they be listed as subentries under their parent forms? In the case of compounds, under which of the components of a compound will the compound be filed? Idioms and other frozen expressions may be filed as subentries or in special bands reserved for these expressions. Derivatives may be easier for the user to find if they are given their own main entries and filed alphabetically among other main entries. But it may be more logical and linguistically more productive to file them as subentries under their respective roots.

Having the entries in computer storage in a structured form, however, mitigates such conflicts somewhat, by allowing you to enter and maintain the file in one form--say the most linguistically logical form--and to re-organize it by program into different forms suitable for other purposes.

Entries are generally keyed to forms, whether root, stem, or word, rather than to meaning or other property. (There do exist semantically organized dictionaries, but under each semantic heading the entries are still filed by form.) Hence the general principle, "one form one entry; different form different entry". Deviations from this principle are caused by homonymy and variant forms. Homonymy may give rise to "one form different entries", that is, the headwords of several entries may be identical. Separate entries may be established because the several meanings of the same form are radically different or because more than one etymon is represented. Some dictionaries adhere strictly to a "one form one entry" policy, and do not accord homonyms separate entries, but most do attempt to distinguish homonyms and place them in distinct entries. In this case, typically a superscript numeral or other distinguishing tag is attached to the several otherwise identical headwords. In entering homonyms into the computer, a similar device may be used: the form may be followed by a dollar sign '$', followed by a number. Thus, formally, the "one form one entry" principle still holds, since the $1, $2, etc. are formally part of the headwords, and distinguish them from each other, e.g.,

```
.hw   bow$1
      ---
.hw   bow$2
      ---
.hw   bow$3
      ---
```

When preparing for publication, the $1 etc. can be converted to an inferior or superior numeral.

The situation of "different forms one entry" arises when there are variant forms--for dialectal or other reasons, which all share the same definitions and other information--they in fact share the same entry. One of the forms should nevertheless be chosen as the headword of the entry, and the others should be listed in one or more "variants" bands.

Cross-reference entries are short entries used only to refer the reader to a "full" entry where more complete information is given. The cross-reference entry may be for a variant, a dialect form, irregular or suppletive form, derived form, etc., which is not itself accorded a complete entry but is mentioned in the full entry for another form. The cross-reference entry is placed in its proper alphabetical position in order that the reader can find it. It consists minimally of the headword itself and a cross-reference band containing the headword of the entry the reader should look up, but may also contain a short gloss and other brief information. If planned for properly, it need not be entered separately into the computer file, but may be constructed by program from specified pieces (e.g. bands) in the full entry. Thus, from an entry containing

```
.hw   adi
var   edi
   ---
```

where the VAR band contains a variant form, a program can construct a cross-reference entry

```
.hw   edi
see   adi
```

which consists of a headword band and a SEE band. Of course, any additional bands may be copied into the generated entry from the original entry. The amount of information to be duplicated in a cross-reference entry is the subject of policy decisions by the lexicographer. This cross-reference entry, along with all the others so generated, can be added to the dictionary, alphabetized in their correct places.

Subentries can be another possible source of generated cross-reference entries. From the subentry for mabea, under the main entry bea,

```
.hw   bea
   ---
..pl   mabea
   ---
```

a cross-reference entry may be generated for mabea, and alphabetized in its proper place, that directs the reader to look for the full subentry under bea:

```
.plg   mabea
plof   bea
```

Again, additional bands may be duplicated in the cross-reference entry, if desired.

Note that the band names in the above examples have been chosen more or less arbitrarily, for the (hypothetical) user's convenience: they have no special meaning for any of the existing programs.

Another question of policy touching on headwords and entries is what representation of a form to use for the headwords. If a morpheme never occurs uninflected, which inflection should be used for citing it as a headword? E.g., verbs in Latin are given in their infinitive, in Sanskrit in a third person form, etc. Should the headword be given, instead, as a bound form, with a hyphen indicating an obligatory inflection? In Bantu languages, should the headword be given with or without the concord prefix? In addition to such morphological choices, there are also phonological choices. Should the form be given in phonetic, in phonemic, or in morphophonemic transcription? Perhaps an orthography which is none of the above is preferable? Yet another dimension of choice is

presented by dialect variation. Which dialect will be taken as the "standard", whose forms are given the full entries, while the "non-standard" forms will be given short cross-reference entries or will only be mentioned in the full entries?

The format conventions cannot, of course, help you answer these questions of policy, but they can, if used properly, provide some flexibility for conversion to a different organization if one set of policies is found to be unsatisfactory.

3.3.2 BAND NAMES, THEIR STRUCTURE AND FUNCTION

Band names may consist of any letters and numbers and special characters except for blanks, commas, and semicolons. They must begin with a letter. They may be in caps or in lower case, but the programs generally do not distinguish between upper and lower case band names. That is, they consider 'DEF' and 'def' to be the same band name. User-written control modules may, however, take into account the difference. One may take advantage of this situation by using upper-case band names for one kind of bands (e.g., bands containing material in the target language) and lower case band names for other bands (e.g., bands containing English).

Three band names, TITLE, FILE, and LIST, have special uses, described in Section 3.3.3, Uses of bands.

One may build internal structure into band names, as in ENG-DF for an English definition band and FR-DF for a French definition band. However, again, none of the existing programs recognize structure within band names. That is, no existing control module could pull out all bands with names containing -DF, say. If such a feature were desired however, one could easily write a function to do so.

Band names usually can be considered to represent relationships. A band name gives the relationship between the information in the band and the headword of the entry. For instance, the band name PS claims that the information in the body of the band is the "part of speech" of the headword, or the band name BSK is an abbreviation for the relationship "technical meaning (of the headword) in basket-weaving technology". Of course, since not all entries will have a BSK band, the band name itself carries information about the headword, namely that the headword is used with a specialized meaning in basket-weaving. Another common way of handling this situation uses mode numbers (see example in 3.3.6, below).

If a band name represents a relationship with the headword, then what relationship does the headword band name represent? The name of the headword band is often simpy HW, carrying little, if any, information, since the period preceding it already marks this band as the first band of the entry and hence the headword. The HW in fact does not represent a relation, but only marks (redundantly) the

body of the band as the headword of the entry. Since this is redundant, one may assign some other, more useful, band names to the headword band. For instance, it may be used to label the dialect of the headword, to give the morphological status of the headword (stem, prefix, full form, etc.), to give the source of the entire entry, etc. Thus instead of HW for every headword, one would have different band names for the different dialects, or morphological types, or sources, of the headwords. Another use of the headword band name would be to label the type of transcription in which the headword is given: phonemic, orthographic, morphophonemic, etc., as suggested in 3.3.3.

3.3.3 USES OF BANDS

Three bands, TITLE, FILE, and LIST have special meanings when encountered in certain situations. TITLE, when used as a main headword band, i.e., preceded by a single period, causes the LISTGEN control module and every control module that produces a listgen-format (l-format) output to issue a command to go to the top of the next page and to use the body of the band as the running top title from then on. .TITLE bands could be manually inserted at specific points in a file to create breaks in the LISTGEN printout. This band is inserted automatically at certain points by the TOAST control module SORTENT, q.v. As a non-main-headword band, TITLE has no special significance for any program.

The FILE band, when occurring as a main headword band, i.e., preceded by a single period, is recognized by two special SPITBOL programs on the PC, called SPLIT and SPLITQ, which are technically not part of the system since they are neither BIRDBATH or TOAST control modules and have nothing particular to do with dictionary files. SPLIT copies sections of a file into other files specified by the body of the .FILE band. All records following the .FILE band, up to the next .FILE band (or until the end of the file), are copied into the file whose name appears as the body of that band. This program is useful for splitting up a long file at specified points into several smaller ones. The SPLITQ program does the same, except it prompts for the changing of a floppy disk for each output file. When not occurring as a main headword band, FILE has no particular significance except as described in the next paragraph.

On the PC, the LIST band can be used in conjunction with FILE bands to form a "list file" which lists the names of other files to be processed in sequence as if they formed a single file. The form of the list file is a single entry with the headword band .LIST followed by FILE bands each containing the name of a file. (This use of the FILE band is different from, and does not conflict with, its use as a headword band as described in the previous paragraph.) For example:

```
.LIST     Koryak file list
FILE      KORA.LEX
FILE      KORB.LEX
FILE      KORC.LEX
      etc.
```

When a BIRDBATH or TOAST run gets such a file as input, rather than processing it as a normal lexical file it processes the files named in it instead (having first checked that it can find them all). Other details of the list file format are these: the .LIST band must be the first band in the list file, and the body of the band can contain anything; bands other than FILE may be interspersed among the FILE bands. They will be ignored, and thus could be used for comments. The occurrence of LIST bands in the lexical files themselves have no effect. The list file facility is not available in the mainframe version.

Apart from these three special bands, bands are normally used to hold various kinds of information about the headword. This subsection contains a catalog of some of the possible kinds of information. Some items in this catalog however do not fit this description. A band can, for instance, be used to segment an entry, or to force a paragraph break for typesetting purposes. Such uses are nevertheless included. No claim is made about the universality, completeness, mutual exclusivity, etc., of this list. Some of the categories may well overlap, and some may not be sufficiently differentiated. Also, not all categories will be of equal relevance to all languages. For discussions of some of these kinds of lexical information please consult the references given in Section 1.2.2e.

We first dispose of two "pseudo-categories" of lexical information. The first is "headword", often assigned to a HW band. Since the headword must be written in some sort of transcription (phonemic, orthographic, etc.), the name of that representation would be a more informative band name than HW. The last paragraph of 3.3.2 mentions other possiblities. For convenience and out of habit, however, we will continue to use HW in examples.

Another common "pseudo-category" of information is "cross-reference". Often, such a band would be established and given a band name such as CF but be put to use for a wide variety of types of cross-reference. In one instance the cross-referenced item may be a dialectal variant of the given headword, in another it may be the root under which a fuller description of the given headword may be found as a subentry, in yet another it may be a semantically related word which the reader is encouraged to compare, or it may be a suppletively related form, etc., etc. Since a cross-reference is not randomly established but is based on some relationship (such as those just listed) to the given headword, a good use of the band format would be to explicitly indicate each such relationship by means of a distinct band name rather than to conflate them all under a general name such as CF.

Furthermore, a pair of mutual cross-references which are logically not symmetrical should have different band names. For instance, the example in Section 3.3.1,

```
.hw   adi
var   edi
   ---          (full entry)
.hw   edi
see   adi       (cross-reference entry)
```

should not read

```
.hw   adi
var   edi
   ---          (full entry)
.hw   edi
var   adi       (cross-reference entry)
```

The first entry refers to edi and the second to adi for different reasons.

Here follows a skeletal listing of some types of lexical information that it would be appropriate to assign to different bands. They are grouped informally under linguistic and other rubrics.

ORTHOGRAPHIC: spelling form of headword; orthographic variants ("judgment" and "judgement"); regional variants ("color" and "colour"); orthographic combining forms (e.g. in English, the combining form shows whether the last consonant is doubled before -ed and -ing).

PHONOLOGICAL: phonetic shape; phonemic shape; phonological variants; phonological characteristics (onomatopoeia, ideophone, expressive, etc.); phonological comments.

MORPHOPHONEMIC: morphophonemic, underlying, or base form; morphophonemic characteristics (sandhi forms, liaison, etc.); morphophonemic comments.

EXISTENTIAL status: "unattested" or "inferred form", hapax legomenon, nonce word, or rare.

MORPHOLOGICAL: morphological status (prefix, suffix, bound root, etc.); root of the word; morphological analysis (if form is morphologically complex); morphological form-class (e.g. conjugation class, gender, concord class, etc.); gaps in the paradigm; alternate forms occupying the same slot in the word's paradigm; allomorphs ("principal parts", ablaut forms, irregular paradigmatic forms, suppletive forms, etc.); derivatives by productive processes which are listed with no further comment (e.g. English un-, -ness, Austronesian causatives, etc.); comments on morphology.

SYNTACTIC: syntactic function class (e.g., part of speech); case-frame or valence formula; syntactic (or semantic) features (animate/inanimate, mass/count, special types of verbs, e.g., of belief, etc.); restrictions on privileges of occurrence; government of, e.g., prepositions; comments on syntactic function (especially for function words, for which including a "definition" band would be inappropriate).

COLLOCATIONAL: characteristic agents, instruments, verbs, manners, etc. (lightning strikes, tide ebbs); common collocations; idioms and idiomatic constructions using the word in a prominent position, with explanations; proverbs and sayings containing the word.

DISCOURSE: comments on any special discourse properties or function of the word.

LEXICAL: synonyms; antonyms; other members of a closed lexical system to which the word belongs (e.g. growth stages of coconut or fish); classifiers and counters; lexical "functions" or "relations" (see the Grimes and Mel'chuk references in Section 1.1.2e); easily confused forms (phonologically similar forms: "caution! do not confuse with...").

SEMANTIC: translational equivalent; description (where translation not available or sufficient); explanation of usage (especially for function words); placename; definition in same language ("monolingual" definition); definition in a lingua franca of the area; definition in another major language commonly used in the area; simplified definition for school edition; literal morpheme-by-morpheme meaning, if form is complex; semantic field and subfield; scientific designation (Latin name) of flora or fauna; ethnosemantic analysis or features relevant to selectional constraints; semantic features; specialized meaning in some technical field ("subject labels"), e.g., military, arts, navigation; semantic relationship (specialization, metonymy, etc.) with another sense of the word; comments.

BACKGROUND information: cultural; historical; "encyclopedic".

ILLUSTRATIONS: phrase or sentence with translations and comments; reference to place in a text for an illustration, which is not quoted at length.

SOCIOLINGUISTIC ("status labels"): marked speech level (formal, royal, literary, Biblical, ritual language, slang, vulgar, obscene, baby-talk, non-standard, substandard, etc.); regional and social-class restrictions; historical: obsolete, archaic; comments.

HISTORICAL and COMPARATIVE: etymology; cognates and related forms in other languages; reconstructed form in parent language; dialect variants; equivalent (in meaning) forms in other dialects; information about borrowing: source language, source word, meaning

of source word; historical development of meaning and usage; neologism; comments.

REFERENCES: to reference grammar, texts, or other works containing information relevant to the word.

ENTRY SEGMENTATION: Special bands may be designated to explicitly mark the beginning and/or end of a particular section of an entry. For instance a section of bands giving cognates in other languages might be delimited with the bands "cog" and "endcog", or a section of bands giving a paradigm, such as stem sets for an Athabaskan verb, may be delimited with the bands "sets" and "endsets". The delimiter bands themselves may be left empty.

TYPOGRAPHICAL: When preparing a dictionary for publication, one may want to specify that certain very long entries should be broken up, for a better typographical appearance, by starting a new paragraph at chosen points in the entries. These points can be marked by inserting a special band invented for this purpose. Also, a space for a picture may be reserved by inserting a special band which gives the dimensions of the space to be left blank. The program that prepares the publication copy can be designed to look out for these bands and to act accordingly.

HOUSEKEEPING: source (informant name, manuscript, radio transcript, etc.); field notebook page number; person who entered the information into the computer file; date entered and date the entry was last edited; assessment of reliability of the information in the entry; queries, points requiring further investigation; notes and queries for other workers on the project; other "private" notes.

3.3.4 THE SEQUENCE OF BANDS AND ITS SEGMENTATION

It is customary to put bands in a consistent order. That is, once a particular order of bands is established, it is normally observed in every entry (except that groups of bands may be repeated if you used mode prefixes). However, consistency is not required and none of the programs check for it, so the order of bands may be altered in some entries if that serves some purpose.

As mentioned under Entry Segmentation in Section 3.3.3, if there is a subsequence of bands that forms an independent section, a pair of special bands can be designated to explicitly mark the section's beginning and end. While none of the existing control modules recognize this convention (they would not treat these bands in any special way), a specially written control module could easily be made to recognize them. This method of segmentation should not be used for repeated, or repeatable, groups of bands, for which the mode notation is available (see 3.3.6).

3.3.5 STRUCTURE WITHIN BANDS

Within the body of bands you may establish further regular structure. Although no existing control module (except INVERT and INVERT2) looks at internal structure within bands, user-written control modules or user-written instructions in BANDAID-type control modules may. Typical band-internal structure consists of variable-length fields separated by delimiter symbols. For instance, in illustration bands there are typically two fields, the illustration sentence itself and the translation. In the examples of such bands in Chapter 2 and Section 3.2, the fields were separated by the vertical bar '|' symbol. Sometimes two translations are given, in two target languages, e.g. Spanish and English, in which case there would be three fields. The same symbol may be used again to separate fields two and three. Another example of band-internal fields is a loan band, in which three pieces of information may be given: the source language, the source word, and a brief gloss of the source word. Often there is no need to introduce any special field-delimiting symbols. That would be the case where the normal punctuation is sufficient to unambiguously delimit the fields.

3.3.6 USES OF MODES.

The basic use of modes is to group together two or more occurrences of bands that belong together in some logical way. All the bands in a contiguous block which carry the same mode prefix are considered to belong together. A typical use of a mode in an entry is to group together all the bands relating to the headword as used in a particular part of speech, separating them from bands pertaining to other parts of speech. Figure 3.3 showed an entry with three groups of bands segregated according to three parts of speech: "NOM", "VB", and "QUAL". Bands can also be segregated according to other properties, e.g., sense, dialect, technical field, source of information, classifier selection, etc., etc.

```
ps    V
1df    to soak
1il    ...  (illustration sentence)
2tech  bsk
2df    to ret (of reeds) in flowing water
2il    ...   (another illustration)
```

In this example of part of an entry, the PS band has no mode prefix, and "belongs" to the entire entry. The first DF and IL bands belong together, as do the following TECH, DF, and IL bands. The latter group of bands pertain to the use of the word in the context of basketry. Presumably the information in the first group is not thus restricted. Another way to treat the restrictive label (bsk) would be to use it as the band name of the definition band instead of putting it in a TECH band:

```
2bsk   to ret (of reeds) in flowing water
2il    ...  (illustration)
```

This is the kind of representation mentioned in 3.3.2.

Another use of modes is to separate out the fields of a band into separate bands, when the fields threaten to become too complicated, or for some other special reason. These bands would have the same mode prefix, to show that they belong together. Illustrations may be treated in this way, with the illustration itself in one band and the translation in another. In fact, it must be done in this way, rather than with a single band, if the words in the translation will be starred for the purpose of finderlist generation (see Appendix I, paragraph r). For example, instead of having two illustration bands like this:

```
il   qarur sara saktway // I *think I'll go tomorrow / Estoy
  *pensando ir ma$nana
il   janiw sasawsi // They *said no / *Dijo que no
```

which may cause undesirable results in the finderlist (INVERT) output, they should be broken into three bands each:

```
1il   qarur sara saktway
1tre  I *think I'll go tomorrow
1trs  Estoy *pensando ir ma$nana
2il   janiw sasawsi
2tre  They *said no
2trs  *Dijo que no
```

and the TOAST control module INVERT2 (see in Section 3.5) should be used to generate the finderlist.

A final use of modes is to set off an "internal" subentry. Recall that upon finishing a subentry (an entry starting with a two-dot headword band) you cannot return to the main entry--you can only start a sub-subentry, another subentry, or another main entry. That is, you must finish the main entry before going down into a subentry. But occasionally one may wish to insert a "run-on" or "internal" subentry within a main entry, continuing with the main entry after the end of the subentry. The effect of this can be accomplished by omitting the dots that signal the beginning of a subentry, and grouping the bands of the "subentry" together by assigning them a mode number. The following example shows an internal subentry for the expression ahu=whenua embedded not only within a main entry but within a mode (mode 1) of the main entry:

```
.hw  ahu
ps   vt
1df   tend, foster, fashion
1il   Na Tuparimaewaewa, nana i ahu mai, ka kiia he tangata.
1il   Ka ahuria o iwi matariki e te rau e pae.
11expr  ahu-whenua
11df  cultivate the soil
11il  kua marara nga tangata...
2df   treat with
2il
   etc.
```

Not all bands in an entry need to belong to some mode, of course. A pronunciation band, an etymology band, a comparative/historical band, a "private" notes band, say, might not have any mode prefix on them, and would typically be placed some at the beginning of the entry (after the headword band) and some at the end. Normally there would not be any reason to intersperse "top level" bands among bands with mode prefixes, although if such a need should arise, the format does allow it.

If bands within a mode need to be further sub-grouped, then submodes can be used--by prefixing two digits to the band names, as in Figure 3.4. Sub-submodes to any depth are possible, by prefixing more digits.

If more than 9 modes are encountered, do not continue with 10, 11, etc., since two digits denote a submode. Instead, start with 1 again. The programs do not check that the digits are increasing, but only detect when they change, from one band to the next, in order to find the groups of bands.

Note that the repetition of a single band does not need to be accompanied by mode numbers. For instance, if there are several illustration bands, they need not be labeled 1il, 2il, 3il, etc. since this would simply be labeling "groups" of one band each. A sequence of simple band labels, il, il, il,..., would be sufficient. Mode numbers do not have the purpose, primarily, of numbering bands or groups of bands; they serve primarily to segregate groups of bands.

Finally, it might be noted that this system of grouping bands with numbers is not ideal, and is under certain (not common) circumstances actually rather cumbersome. It is, however, easy to remember and does have some advantages from the point of view of programming.

3.3.7 USES OF SUBENTRIES

Subentries (whose headword band label begins with two, or more, dots) are typically used for derived words. The subentry is considered to be attached to the main entry, and not an independent

entry. However, all of the usual apparatus of bands and modes is available for subentries. In particular, mode numbering in the main entry does not carry over into the subentry; it begins over again in each subentry. There may be any number of subentries following a main entry. The subentries may be placed in any order desired; the format does not require that any particular principle of ordering be followed. They are carried along with the parent entry in exactly the order in which they were entered, unless the file is processed by a subentry-sorting control module.

Each two-dot subentry may be followed immediately by one or more sub-subentries (with three dots), before the following subentry is begun. Sub-subentries are typically used for words derived from the headword of the immediately preceding, parent, subentry. The tree of subentries and sub-subentries, etc., may extend to any desired depth.

We will however, continue to use the term "subentry" to refer to subentries and sub-subentries etc. of whatever depth.

The notation allows levels to be skipped, e.g. going directly from a 2-dot subentry to a 4-dot subentry. A plausible interpretation of such skipping is hard to discover however, so normally there is no reason for doing this.

Compounding, as opposed to processes involving only derivational morphemes, creates words which can logically be placed under more than one main entry, since more than one lexical morpheme or word enter into the compound. The decision as to which main entry to place the compound under is a lexicographical decision, but it should be remembered that, whatever the decision, an appropriate program (control module) can make a copy of the subentry, or of a portion of it, and insert it under each of the other relevant main entries.

Subentries are also often used for idioms or other expressions that require more extensive treatment than is convenient to place within a single band. For instance, dialect information, syntactic peculiarities, and illustrations could be recorded in the subentry for the expression. A disadvantage is the property of the subentry notation that prevents you from returning to the parent entry.

It may be desired to place some material at the very end of a tree of subentries, that is, just before going on to the next main entry. For instance, one may want to put a discussion of synonyms at this position. Or one may want to place the illustrative sentences for all the headwords--main as well as subentry headwords--after all of the subentries. Since it is not possible to return to the main entry at this point, one has the choice of placing such end matter in bands in the last subentry, or of opening a new subentry (which would require a vacuous or "dummy" headword band) solely for holding the end matter. The latter choice seems neater. This subentry would be a subentry in form only, since it does not have any derived

word, or any word, as its headword. Its bands would contain the discussion of synonyms, illustration sentences, or whatever, considered to be part of the main entry. In the following schematic entry, the last "subentry" contains all the illustration sentences for the main headword as well as for the subentry headwords:

```
.hw   kirel
   ---
..hw   kekirel
   ---
..hw   el kirel a...
   ---
..ils   (dummy subentry headword band)
il       (illustrative sentences
il        for kirel as well as
il        for kekirel and el kirel a...)
```

3.3.8 SOME GENERAL PRINCIPLES

The "band-by-band principle". Conventions need not always apply to all bands; they may be established on a band-by-band basis: a given convention can be restricted to certain band types. For instance, the asterisk, *, may be used in DF and SEM bands, say, to mark keywords for finderlist purposes, while the same symbol may be used in a historical band, HIST, to mark reconstructed proto-forms. A capital "B" may be used in a phonetics band to represent an imploded b, but have no such meaning anywhere else. In short, each band may be assigned its own set of conventions.

A VERY IMPORTANT COROLLARY of this principle is that you should maintain a written record of these band-by-band conventions, and make sure that everyone on the project understands and follows them. A certain amount of consistency-checking can be done by program, and the BANDSORT control module generates an output which helps the user detect inconsistencies (see 3.7). Obviously it is best to be consistent to start with. It is advisable to keep a loose-leaf notebook, a "codebook", in which each band is assigned a separate sheet on which are recorded the conventions for that band.

Segregate different types of information into separate bands, to the extent that that is comfortable. The discussion of "pseudo categories" at the beginning of 3.3.3 provides some examples of this exercise. First-time users of the format tend to underdifferentiate their bands. However, information that is logically differentiated is easier to work with and potentially more useful, often in unforseen ways. It may take some experimentation to settle on a system of bands that is at the same time usefully differentiated and manageable in size. Typical dictionaries processed in the past have used from 20 to 50 bands. You may, of course, add a new band whenever one is needed; the programs are indifferent to what, and how many, bands you have.

Try to keep each band simple. If the structure of information within a band threatens to become too complicated, try to simplify or see if some of the structure can be shifted out into more than one band or a mode group or a subentry.

Include more information rather than less. A subtle example of this is not to automatically capitalize the first letter of definitions. Only capitalize it if the first word is a proper name. In this way more information is captured. An effect of this is that keywords and phrases pulled into the finderlist will not appear to be randomly capitalized. For publication purposes it is always possible to automatically capitalize the beginnings of all definitions. Another example is to anticipate font changing when the dictionary is photocomposed, by inserting unpredictable font shift codes in the body of the bands, e.g., so that Latin names embedded in definitions or other bands will be printed in italics. It is easier to put these shift symbols in at the start than to edit them in later.

On the other hand, avoid encoding redundant information. If a band will always begin in say, italics, then it is not necessary to place an italics shift symbol at the beginning of the band. Or if all occurrences of a certain band are thought of as being enclosed within parentheses, then it is not necessary to actually type the parentheses, since that is a predictable feature for this band. They can be added easily by a program when the file is prepared for publication.

In choosing among several schemes of organizing a dictionary using the band format conventions, keep in mind that the particular scheme chosen need not be the only form that the dictionary will be able to take. If the scheme is properly designed, it should always be possible to re-organize the dictionary automatically into any of the other schemes contemplated. Attention should therefore be paid to choosing a scheme that preserves the most information, and that represents it in the most natural and least redundant way. These are fortunately often mutually supporting criteria. Such a scheme tends to be also the easiest one to start from to generate the other organizations.

3.4 BIRDBATH CONTROL MODULES

Control modules (see Section 3.1) are the programs that actually operate on the data. This section describes those used with BIRDBATH, which does not do any sorting. Section 3.5 describes those used with TOAST, which does sort. These sections decribe only what the modules do and not how you use them. Since their usage can vary somewhat from installation to installation, this information, in the form of "recipes", is segregated in another chapter, Chapter 4.

As suggested in Chapter 1, this set of control modules is extremely limited. They might be viewed as a sampling of what could be written. If you do not find a control module in this or the next section that meets a particular need, it may be possible to write a new one or to modify an existing one. Also, the set of control modules gradually changes. New ones are added, and existing ones removed or absorbed into new ones. This manual may not always accurately reflect the status of the actual programs that you have.

The description of each module includes descriptions or mentions of the SPECS parameters (see Section 3.1) relevant to the module. Certain common parameters, in particular the families of Input, p-format, and l-format parameters, are not described repetitively, but are only mentioned where relevant. Descriptions of these families, and descriptions or references for all the other parameters as well, are to be found in Section 3.6.

Most of these control modules read an input data file and produce an output data file. The input file must be in p-format (see 3.2.1g) for a control module to be able to read it. The output data file may or may not be in p-format, depending on the control module and on the setting of the PFORMAT parameter. If the description of the control module states that this parameter is available for setting, then the module will, with PFORMAT set to 1 (or any non-null value), produce a p-format output file. Otherwise its output is in some other format. Only p-format files can be used as input to the programs.

All control modules also produce a log file or printout, chronicling significant stages in the run and giving error messages from the program. In the mainframe version, the run log is normally a printout; on the PC it normally goes into a file named MESSAGE. The log should be checked after every run. Some control modules, such as STATS, also place their regular results in the log.

a. BANDAID

This module produces an edited output file from an input dictionary file. The editing is done by 'BANDAID-type' instructions supplied by the user in the SPECS file. These instructions are statements in SNOBOL4 or Spitbol. They specify what is to be done

to each band, on a band-by-band basis, and must follow certain conventions, illustrated in the example in Fig. 3.6. All the bands for which BANDAID instructions are given must be mentioned in a BANDS = '...' statement. The statement(s) for each band must begin with a label consisting of the band name (in caps regardless of whether the band name in the data is in caps or lower case) followed by a period. Each BANDAID instruction should lead to a RETURN or to SKIP. If it goes to RETURN, the band is output; if to SKIP, the band is not output. Bands not mentioned in the BANDS parameter, and which presumably have no corresponding BANDAID instructions, are simply written out without change, unless the NOWRITE parameter is set to a non-null value, in which case they are not output. Control must skip around the BANDAID instructions (see the go-to :(EOSPECS) in the example below). The BANDAID instructions may use any of the public variables and built-in functions (except the input functions). The public variables pertaining to the current band are HLEV ("headword level", containing the zero or one or more periods on the current band label), MODE (the mode prefixes, if any, on the current band label), BAND (the current band name), and BODY (the body of the current band). The output is normally in LISTGEN format (l-format), and all the parameters relevant to LISTGEN are available (see LISTGEN control module). (Thus, this BANDAID control module supersedes the old EDLIST control module). The output can alternatively be placed in p-format, for further computer processing. To do this, set the parameter PFORMAT to a non-null value. Note that previously this was the default for BANDAID; now it is not. (P-format and l-format are described in 3.2.1g.)

The SPECS file for a BANDAID run might look something like the following:

```
    PFORMAT = 1
    BANDS = 'HW,PR,ETYM,DF'                          :(EOSPECS)
HW.   BODY  '$' ANY('1234567890')  =                 :(RETURN)
PR.
ETYM.                                                :(SKIP)
DF.     BODY   '*'  =                                :S(DF.)
DF.1    BODY = REPLACE(BODY,'¬',' ')                 :(RETURN)
EOSPECS
```

Figure 3.6

In Figure 3.6, the statements specify that the output is to be in p-format. The bands for which there are statements below are given in the BANDS parameter (HW, PR, ETYM, DF). Note the skip around the BANDAID instructions. The BANDAID statements specify that

1) In HW bands, '$' and the following digit will be deleted.
2) PR and ETYM bands are not to be output.
3) In DF bands, asterisks will be deleted and '¬' will be replaced by a space.

The only required SPECS parameter needed for a BANDAID run is BANDS (followed, of course, by the BANDAID instructions), and even that is not necessary if the only purpose of the run is to listgen the entire file (but then LISTGEN would normally be used for that). Other possible parameters are NOWRITE, PFORMAT, the Input parameters, and the p-format or l-format parameters, depending on whether PFORMAT is set or not.

b. BANDFILT

This control module "filters" the bands of an entry according to several criteria. First, the headword band is always passed. Next, the first band on a "priority list" of bands that is found in an entry is passed. This list is given by means of the SPECS parameter BANDPRIORITY, in the same format as BANDS. Finally, any bands specified by means of the system of four mutually exclusive SPECS parameters, BANDS, NOTBANDS, ALLBANDS, and NOBANDS, are passed, as follows:

BANDS	=	'...'	pass the bands in the list
NOTBANDS	=	'...'	pass all bands not in this list
NOBANDS	=	1	pass no bands
ALLBANDS	=	1	pass all bands (the absence of all four parameters has the same effect)

Every entry and subentry has something, at least the headword band, output by BANDFILT.

The usual Input parameters, and PFORMAT and either the p-format or the l-format parameters, are available. Output is normally in LISTGEN format (l-format). If PFORMAT is set to a non-null value, the output will be in p-format. (P-format and l-format are described in 3.2.1g.)

c. BANDINV

This module has been superseded by the STATS control module, q.v.

d. BANDPACK

This module "packs" all the bands in an entry or subentry into a single paragraph and outputs the paragraph. It normally packs only the bodies of the bands, not the band labels. Exactly what is packed for each band, however, can be controled by BANDAID-type statements in the SPECS file (see BANDAID control module, above), operating on the BODY variable, which holds the body of the current band. Bands may be skipped (not packed) by going to SKIP instead of RETURN, just as in BANDAID. Bands not mentioned in the BANDS = '...' statement will be ignored. However, if the BANDS parameter is not assigned anything, all bands will be packed. The output

normally is in LISTGEN format (l-format), with one paragraph per entry (considered the "headword" band). Since the band label for this "band" will be superfluous, it should be suppressed with a SPECS statement, SUPBANDLABEL = 1 . Also a statement SUPHEADING = 1 should be given. Output may be put in p-format by setting PFORMAT to a non-null value.

In summary, the SPECS parameters available for BANDPACK are BANDS, PFORMAT, the Input parameters, and the p-format or l-format parameters, depending on whether PFORMAT is set or not. In addition, for l-format output, the l-format parameters SUPBANDLABEL and SUPHEADING should be set. No parameters are required. (P-format and l-format are described in 3.2.1g.)

e. EDLIST

This has been superseded by the new BANDAID control module, q.v.

f. EDLISTB

EDLISTB is not available in the PC version, where the more versatile control module ONBANDSR should be used.

This module outputs only main headwords plus any specified bands that satisfy a given pattern. If none of the specified bands occurs in an entry, or if the given pattern does not occur in one of the specified bands in an entry, then nothing is output for that entry. The bands must be specified in the SPECS file with a statement such as the following:

```
BANDS = 'df,il,pr,etym'
```

If a pattern is not specified, all the specified bands will be output. If one is specified, only those specified bands that also satisfy the pattern will be output. The pattern must be assigned, in a SPECS statement, to the variable PATTERN . Some examples of possible patterns are:

1. empty bands:
```
PATTERN = POS(0)  RPOS(0)
```

2. bands containing either the string 'cat' or the string 'dog':
```
PATTERN = 'cat' | 'dog'
```

3. bands consisting ONLY of the string 'cat' or the string 'dog':
```
PATTERN = POS(0)  ('cat' | 'dog')  RPOS(0)
```

4. bands in which there is a '%' not followed anywhere in the same band by a '|', '&', or '%':
```
PATTERN = '%' ARBNO(NOTANY('|&%')) RPOS(0)
```

5. dfeng bands which contain 'dog' and dffr bands which contain 'chien':

```
PATTERN = *IFBANDS('dfeng') 'dog'  |  *IFBANDS('dffr') 'chien'
```

The output is normally in LISTGEN format (1-format). If p-format is desired in order to produce a file that can be further processed, then set the parameter PFORMAT to a non-null value in the SPECS file. (P-format and 1-format are described in 3.2.1g.)

In summary, the SPECS parameters available for EDLISTB are BANDS, PATTERN, PFORMAT, the Input parameters, and the p-format or 1-format parameters, depending on whether PFORMAT is set or not. At least BANDS should be specified, otherwise no output would be generated.

---

g. LISTGEN

This module outputs a dictionary file in the so-called 'LISTGEN format' (1-format), which includes indentations and line skips to make a dictionary in band format more readable (see, e.g., Fig. 2.11). (P-format and 1-format are described in 3.2.1g.) The SPECS parameters mostly concern the amount of indentation in various situations and other such formatting matters. If no SPECS statements are given, certain default values are assumed. SPECS statements may be used to override these values. In the following list of LISTGEN parameters, the default values are given on the right of the equals sign following the parameter name. The default value for TITLE is a blank line. Another feature which the LISTGEN control module provides is printing the line numbers with each paragraph, if the input file contains line numbers. This feature can be suppressed by setting SNUM to a non-null value by means of a SPECS statement.

LMARGIN = 0 Character position of the left hand margin.

RMARGIN = 132 Character position of the right hand margin (on mainframe version).
= 79 (on PC version).

TITLE = to provide a title to be printed on each page, supply a title in a statement:

TITLE = '....'

(This can be overridden by the occurrence of TITLE bands in the file itself: see 3.3.2.)

LINESPERPAGE = 60 Lines per page. Set this to 80 when using 11-inch paper at 8 lines per inch. For 11-inch paper at 6 lines per inch (the normal case), 60 lines is fine.

HEADHANG = 10 Number of characters that headword bands overhang the other bands in an entry or subentry.

SUBENTINDENT = 3 Number of characters of indentation for each

level of subentry.

SUBMODEINDENT = 2 Number of characters of indentation for each level down of modes.

In summary, the SPECS parameters available for this module are SNUM, the Input parameters, and the l-format parameters, most of which are described above (a few more obscure ones are listed under "l-format parameters" in 3.6). No parameters are required.

h. NULLRUN

NULLRUN is not available in the PC version. This control module produces a "null run". It does not read any data. It can be used to exercise the initializations (see Section 6.2.1.1) or to generate a compilation listing of the built-in functions. No SPECS parameters are relevant.

i. ONBANDS

ONBANDS is not available in the PC version, and is being phased out in the mainframe version. The more versatile ONBANDR should be used.

This module outputs those entries (entire entries) that contain specified bands. That is, if an entry contains one of these bands, then the entire entry is output. The names of these bands are specified in a list, assigned to the SPECS parameter ONBANDS, e.g.:

```
ONBANDS = 'df,cf,phr,il,ps'
```

Furthermore, if a SNOBOL4 pattern is assigned to the parameter ONPAT the succesful matching of the pattern in the bodies of the specified bands becomes an additional criterion for selection. That is, an entry must have one of the specified bands, and the pattern must match in the body of the band, before the entry is selected.

The output is normally in LISTGEN format (l-format), for printing. If the output is to be stored in p-format, for further processing, set PFORMAT to any non-null value, e.g. PFORMAT = 1 .

In summary, the SPECS parameters available are ONBANDS, ONPAT, PFORMAT, the Input parameters, and the p-format or l-format parameters, depending on whether PFORMAT is set or not. At least ONBANDS is required, otherwise no output would be generated. (P-format and l-format are described in 3.2.1g.)

j. ONBANDS2

* ONBANDS2.SPT RETRIEVES ENTIRE NEST OF ENTRIES IF ANY BAND IN IT * SATISFIES ONBANDS AND ONPAT. SPECS PARAMETERS AVAILABLE: PFORMAT, * AND THE LISTGEN PARAMETERS LIKE RMARGIN. NOT AVAILABLE ARE * THE BAND SELECTION PARAMETERS LIKE BANDS, NOTBANDS, ETC. This control module retrieves an entire nest of entries (main entry and all its sub- and sub-sub- etc. entries) if a specified band, or specified bands, occur anywhere in it and if a specified pattern is satisfied in the body of the specified band(s). The criteria for retrieving an entry are the same as for ONBANDSR, described below, but the scope of how much of an entry is retrieved is different. Whereas ONBANDSR retrieves only the headword bands from entries dominating a target subentry (one that satisfies the search conditions), this control module retrieves the entire nest of entries (main entry and all subentries) that contains the target subentry. Furthermore, every single band is retrieved: there is no way to select the bands. The retrieved entries are placed in the output file, in either l-format or--if PFORMAT is set to a non-null value--in p-format.

k. ONBANDSR

This control module, like ONBANDS, selects entries on the basis of whether or not a given band, or any of a list of bands, is found in an entry, and on whether a given pattern, when such a pattern is given, is satisfied in the body of the band. The bands to be inspected are given by means of the SPECS parameter ONBANDS, and the pattern, if any, is given by means of the parameter ONPAT, just as with the ONBANDS control module.

If the parameter REJECT is set to a non-null value, then the conditions for selection and rejection of entries are reversed from the above description.

The bands to be output with a selected entry are: the headword band, the band(s) satisfying the ONBANDS and ONPAT conditions (unless REJECT is set), and any other bands specified by one of the four mutually exclusive parameters

BANDS	=	'...'	the bands specified in this list
NOTBANDS	=	'...'	all bands except those in this list
NOBANDS	=	1	no other bands
ALLBANDS	=	1	all bands (not specifying any of these four parameters will have this effect too)

Each subentry will also be treated in the above way. In addition, if a subentry or main entry is not itself selected but one of the subentries which it dominates is, then the dominating entry or subentry will be represented in the output, but only by its headword band. In this way, there will always be a proper path to a selected

subentry, whether or not its dominating entry or subentry is selected.

While ONBANDS and ONPAT represent independent conditions, that is, the pattern ONPAT can be satisfied in any of the bands given in ONBANDS, it can also be arranged that one pattern be applicable in certain bands while another pattern be applicable in certain other bands. This can be done simply by concatenating a call to *IFBANDS(bands) (see Section 6.2.4.1) to the beginning of the appropriate pattern and building an alternation of these patterns:

```
ONBANDS = 'A,B,C,D'
ONPAT =  *IFBANDS('A') PA | *IFBANDS('B,C') PBC | *IFBANDS('D') PD
```

These SPECS statements say: 1) the bands to be inspected are A, B, C, and D; 2) look in A bands for the pattern PA, look in B and C bands for the pattern PBC, and look in D bands for the pattern PD. If any of these patterns is found, select the entry.

In addition to the above SPECS parameters, the Input parameters and the output parameters, viz, PFORMAT and p-format or l-format, are available. Normally output is in LISTGEN format (l-format). If PFORMAT is set to a non-null value, it is in p-format.

---

l. SEQCHECK

This control module checks through a file for correct alphabetical order of the headwords. Whenever it finds a pair of consecutive entries in which the second headword is not strictly alphabetically greater than the first, it copies the two headword bands into the output file, where they may be inspected. Correct alphabetical order is defined by a handle function supplied by the user in the same way as for the TOAST control module HANDSORT. It would normally be placed in the "userfns" file, and the name of the function would be specified in the SPECS file with a statement like

```
HANDLENAME  =  'KORHAND'
```

If no handle function is supplied, a default alphabetical scheme is assumed. See the description of HANDSORT in the next section for a little more information about writing handle functions.

---

m. STATS

This module reads through a file and prints out various statistics in the BIRDBATH RUN LOG: the number of times each band occurs, the number of different bands, the number of paragraphs (number of band occurrences), the number of main entries, total number of entries (including sub- and sub-sub- etc. entries), total number of characters in the file. In the PC version, the bands (with their frequencies of occurrence) are listed in alphabetical

order (actualy, computer collating sequence order); in the IBM mainframe version they are not. A BANDS = '...' or NOTBANDS = '...' statement may be given in the SPECS file to restrict the first two statistics. The other statistics are compiled always for the entire file.

The SPECS parameters available are BANDS or NOTBANDS (mutually exclusive), and the Input parameters. None are required.

3.5 TOAST CONTROL MODULES

TOAST control modules perform operations that require sorting or re-ordering of material in the dictionary file. Each module consists of two "phases", one being executed before sorting and one after. In general, separate SPECS statements are required for each phase.

The description of each module includes descriptions or mentions of the SPECS parameters (see Section 3.1) relevant to the module. In the mainframe version, SPECS parameters for TOAST modules are separated into phase-1 parameters and phase-2 parameters, but in the PC version both sets are normally put in the same SPECS file. Certain common parameters, in particular the families of Input, p-format, and 1-format parameters, are not described repetitively, but are only mentioned where relevant. Descriptions of those families, and descriptions or references for all the other parameters as well, are to be found in Section 3.6.

As with the BIRDBATH control modules, most TOAST control modules read an input data file, which must be in p-format, and produce an output data file which may or may not be in p-format. If the description of the control module states that the PFORMAT parameter is available for setting, then the module will, with PFORMAT set to 1 (or any non-null value), produce a p-format output file. Otherwise its output is in some other format. Only p-format files, of course, can be further processed by BIRDBATH or TOAST control modules.

All TOAST control modules produce two log files or printouts, chronicling significant stages during the execution of the two phases, and giving error messages from the execution. Each log is labelled as a BIRDBATH RUN LOG, since each phase is considered a BIRDBATH run. In the mainframe version, the run logs are normally part of the whole printout from the run; on the PC they normally go into two files, MESSAGE and MESSAGE2. The logs should be checked after every run.

There are recipes in Chapter 4 illustrating the usage of TOAST control modules.

---

a. BANDCORD

This control module constructs a simple center-format (also known as a Key-Word-In-Context) concordance of specified bands, in the general format of a bandsort (see BANDSORT control module). Figure 2.13 showed the format of a bandsort. The band concordance format differs from this in that each band body appears once for each word in it, centered on the word. All the occurrences of all the specified bands are sorted by the centered words. Each body is truncated, as necessary, at the left and right edges of the available space. The boundaries of the "word" are spaces.

Currently it is not possible to specify other delimiter characters. The default widths of the concordance fields (and their names) are 35 for the headword field (HWFIELD) and 12 for the left side of the centered concordance field (LFIELD). Ten spaces are reserved for the mode numbers and band name, and the left and right margins are determined by LMARGIN (default 0) and RMARGIN (default 132 on the mainframe, 79 on PC). The bands to be concorded are specified with the BANDS or NOTBANDS parameter. All these parameters are Phase-1 parameters.

b. BANDSORT

A bandsort is an alphabetical listing of bands from a dictionary file. The bands are in alphabetical order by band name, and within each group of bands with the same name the bands are in alphabetical order by the body. The output of this program is in neither p-format nor l-format. (See Fig. 2.13 for an example of a bandsort output.)

In the phase-1 SPECS file, a list of bands that are to be printed or that are to be ignored may be given, by using the BANDS or NOTBANDS parameter, respectively, e.g.

```
BANDS = 'il,phr,ps,df'

        or

NOTBANDS = 'il,phr,ps,df'
```

If no such statement is given then all bands are taken. The HWFIELD parameter specifies how wide a field is reserved for the headword in the BANDSORT output (see, again, Fig. 2.13). This field is at the left of the printout. Any headwords longer than this will be truncated on the printout. The default is 35. (In Figure 2.13 the field is only about 9 characters wide, in order to fit within the figure box.) KEEPSEQ is another parameter available in phase-1 of BANDSORT. Normally, all occurrences of each band are sorted into order by their band bodies, as shown in Fig. 2.13. However, if KEEPSEQ is set to a non-null value, they will be kept in their original relative order. In addition to these parameters, the Input parameters (q.v. in Section 3.6) are available. No SPECS statements are obligatory in phase-1.

In the phase-2 SPECS file may be specified how many blank lines are to be skipped when a new group of bands (with a new name) begins. The parameter is SEPARATION. The default is 1. If some number is given that is larger than the number of lines on a page, e.g. SEPARATION = 70, then every new block of bands will start on a new page. OFFSET is the amount of indentation given to a continuation line in the bandsort output. It is normally set at 50, in order to clear the headword field and band label field. If HWFIELD is reset in phase-1, then it may well be desireable to reset

OFFSET. All the page-formatting parameters (q.v. under '1-format parameters' in Section 3.6) are also available, i.e., all the 1-format parameters except HEADHANG, SUBENTINDENT, SUBMODEINDENT, and SUPBANDLABEL. No SPECS statements are, however, obligatory in phase-2.

---

c. HANDSORT

This module alphabetizes a file by headword, i.e. the body of the first band of each entry. Sub-entries are not disturbed--they are left in their relative order beneath their respective main entries. The sorting is not based directly on the form of the headword and the computer collating sequence; rather, it is based on another form, called the handle, generated from the headword by a function which may be supplied by the user. In the absence of a supplied function a default handle function is used, which causes the upper and lower case distinctions to be ignored. It also ignores non-alphabetic characters. If a more sophisticated alphabetization scheme is desired, a special function must be supplied, and placed in the phase-1 USERFNS. (See Section 4.1, j.) It may be called 'HANDLE', must take one argument--the string for which a handle is to be made, and must return the resulting handle as value. This must, of course, be written by a programmer. If it is not called 'HANDLE', a statement saying what it is must be placed in the phase-1 SPECS file, e.g.

```
HANDLENAME = 'POTHAND'
```

As usual with most control modules, the Input parameters (q.v. in Section 3.6) are also available here, in phase-1. No SPECS statements are, however, obligatory for a HANDSORT run.

The output of this program is the sorted dictionary, by default in LISTGEN format (1-format), in which case the 1-format parameters (q.v. in Section 3.6) are applicable in phase-2. Presumably, however, one would want to save the sorted dictionary in a file for further computer processing or for editing. That means having it written out to a file in p-format. In that case one would set PFORMAT to a non-null value in the phase-2 SPECS file, and the p-format parameters (see 3.6) would be applicable. A phase-2 parameter TRACEHANDLE may be set to a non-null value, during check-out of a new handle function, to cause a list of the sort records with their handles to be printed out. Another phase-2 parameter, SUPDUPTRACE, if set to a non-null value, suppresses a listing of "duplicate handles"--headwords which have identical handles and which presumably one would want to be alerted to. For these words the alphabetical ordering, as specified by the given handle function, is ambiguous, or "underspecified". This listing is given in the BIRDBATH RUN LOG of phase-2. In summary, in phase-2 of HANDSORT, the SPECS parameters available are PFORMAT, TRACEHANDLE, SUPDUPTRACE, and the p-format or 1-format parameters, depending on whether PFORMAT is set or not.

d. INVERT

This module is not distributed with the PC version; the more versatile INVERT2 should be used.

This module generates a finderlist, the elementary characteristics of which have been described in Chapter 2. Due to the complexity and number of conventions pertaining to this control module, the details have been relegated to an appendix (Appendix I) in order to unclutter this chapter. Only the most basic SPECS parameters are mentioned here.

In the phase-1 SPECS file, the band(s) in which the program is to look for asterisks (which mark keywords) must be stated. This is done by a BANDS = '...' statement, e.g.

```
BANDS  =  'def,sem'
```

Such a statement in the phase-1 SPECS file is required for running INVERT, since otherwise the program would not know what bands to look for asterisks in. All the other parameters are optional. A large number of them are provided for controlling the delimitation of keywords and of the phrases that they are embedded in. A detailed account of them is given in Appendix I. In addition, the Input parameters (q.v. in Section 3.6) may be specified in the phase-1 SPECS file.

In the phase-2 SPECS file, no parameters are required. Output is in band format, with two bands, KW and PH . Output is normally in LISTGEN format (l-format) for printing out, but may be put in p-format by setting the parameter PFORMAT to a non-null value. In l-format, since a finderlist has only two bands, and the KW band is always the first band in an entry, the control module suppresses printing of the band labels. If, however, band labels are desired in the printout, set the parameter PRINTBANDLABELS to a non-null value in the phase-2 SPECS file. In summary, the parameters available to be modified in the phase-2 SPECS file are PFORMAT, and the p-format or l-format (and PRINTBANDLABELS) parameters (see Section 3.6), depending on whether PFORMAT is set or not.

e. INVERT2

This is a more general version of INVERT (see above) which 1. constructs, for keywords within subentries, a reference to the main headword as well as one to the subheadword; and 2. looks for keywords (starred words) in "translation" bands associated with "example" bands in the same mode. The example bands are treated as subheadwords. Details are presented in Appendix I, paragraph r. This control module may eventually supersede INVERT if its performance on the subset of INVERT tasks proves comparable to that of INVERT itself.

The phase-1 SPECS parameters are the same as for INVERT, except that two more "BANDS" type parameters are available, one, XBANDS, for specifying the name(s) of example band(s), the other, TBANDS, for specifying the name(s) of translation band(s).

The phase-2 SPECS parameters are the same as for INVERT.

f. NULLRUN

This module is not available in the PC version.

This module produces a "null run". It does not read any data. It can be used to exercise the initializations (see Section 6.2.1.1) or to generate a compilation listing of the built-in functions in BIRDBATH and TOAST. No SPECS parameters are relevant.

g. REBANDLE

This module re-orders the bands in each entry according to a given order. The new order is given in terms of a list of band names in a file BANDLIST in phase-1 of the run. In that file there must be one bandname per line, with nothing else. REBANDLE does not take bands out of their modes: the grouping function of mode numbers is respected.

In the phase-1 SPECS file no parameters are available except the Input parameters (q.v. in Section 3.6). In the phase-2 SPECS file are available PFORMAT, and the p-format or l-format parameters depending on whether PFORMAT is set or not. Output is normally in LISTGEN format (l-format), but may be put into p-format by setting PFORMAT.

h. SORTENT

This module sorts the entries in a file, much as HANDSORT does, except instead of sorting by the headword band as HANDSORT does, it can sort based on anything in the entry. As with HANDSORT, the user must provide a handle function to phase-1 to generate the sort handle for the entries. The name of the handle function must also be given in the phase-1 SPECS file as a parameter HANDLENAME. Unlike the handle function supplied to HANDSORT however, this handle function takes no argument, but bases the handle on other information. This information is put aside by another user-supplied function, which must be called BANDPROC(). This function is executed by the control module each time a band is read. The function may test for the band and do whatever else it needs, and it puts aside information, say by using the SET function (SSET function in the PC version) (see 6.2.4.3). The handle function is called at the end of the entry, and can make use of any information put aside by BANDPROC during the reading in of the entry. The entry is sorted

according to the handle so generated. The handle function may specify that the entry be skipped, and the BANDPROC function may specify that a given band be ignored. See the control module itself for further comments. For phase-2, the p-format parameters are relevant if PFORMAT is set; otherwise the l-format parameters are relevant.

i. SORTSUB

This control module alphabetizes the two-dot subentries under each main entry. The subentries are not taken out of the nest, but are re-arranged (if necessary) into alphabetical order, still under the same main entry. The sub-sub- and deeper entries under a subentry are by default not reordered, but kept as a unit with the subentry. If it is desired to reorder all deeper subentries as well, then the parameter SUBSUB should be set to a non-null value. This is a phase-1 SPECS parameter. Note that the hierarchical relationship in a nest of entries is never altered, only the sequence at each level may be changed. As with HANDSORT, a handle function must be supplied. Follow the conventions described under HANDSORT, above, for writing the function. The name of the function must also be given as a phase-1 parameter HANDLENAME. As for phase-2 SPECS parameters, the p-format parameters are relevant if PFORMAT is set; otherwise the l-format parameters are relevant.

j. XCHECK

This control module checks cross-references to find references that do not occur as main headwords in the file. Cross-references may be in any form in the file, but the user must supply a function XREF(), of no arguments, that looks in the current band and returns a list of (zero or more) cross-reference words separated by commas (and optionaly spaces). The bands in which the function is to look must be given in terms of a list of bandnames in the parameter BANDS. The BANDS = '...' statement must be given in the phase-1 SPECS, and the XREF() function must be given in either the phase-1 SPECS or the phase-1 USERFNS file. A handle function may be provided in the USERFNS file, as for the HANDSORT control module, q.v., for alphabetizing of the output. Thus HANDLENAME is available as a phase-1 SPECS parameter. In addition, the Input parameters (q.v.in Section 3.6) are available in the phase-1 SPECS.

The output is a list of entries consisting of two bands: headword band and a cross-reference band containing one cross-reference word. The bands are labeled with the same names as their bands of origin in the input file. These entries represent unresolved cross-references, i.e., cross-reference words which do not occur as headwords in the input file, along with the main headwords under which they occurred. These entries are in alphabetical order by the cross-reference words, not by headword. If the output is saved in a file in p-format, it can later be sorted into order by the headword

using the HANDSORT control module. Thus in the phase-2 SPECS file, the parameters available are PFORMAT, and the p-format or l-format parameters (q.v. in Section 3.6), depending on whether PFORMAT is set or not. If PFORMAT is not set, the output is in LISTGEN format (l-format), and the l-format parameters are relevant.

3.6 SPECS PARAMETERS

This section contains an alphabetical list of the parameters that can be overridden in the SPECS file when running BIRDBATH or TOAST with existing control modules. With each parameter is given its type (string, number, flag, or pattern), its purpose, and its default value. In each case of a TOAST parameter it is noted whether it is a phase-1 or a phase-2 parameter. This is relevant for the IBM mainframe version but not for the PC, where both types of parameters are given in the same SPECS file. What function uses the parameter is also mentioned for the information of programmers.

If a parameter is to be overridden with a statement in the SPECS file, the form of the statement must be consistent with the type of the parameter. If the parameter is a string, the value must be enclosed in single or double quotes, e.g.

```
BANDS  =  'HW,DF,CF'
```

If it is a number, it must be a simple integer without quotes:

```
LMARGIN  =  10
```

If it is a flag, for all practical purposes any number will do:

```
PFORMAT  =  1
```

If it is a pattern, it must conform to SNOBOL4 pattern syntax:

```
ONPAT  =  'cpd.'  |  'redup.'  |  'tb.'
```

In all cases there must be spaces on both sides of the equals sign.

ALLBANDS a flag used by certain control modules that select bands in addition to certain fixed bands such as the headword band. If set to a non-null value, it causes all bands to be selected. It is mutually exclusive with NOBANDS, BANDS, and NOTBANDS. It is used by the function IFBANDQ(). ALLBANDS is relevant to the control modules BANDFILT and ONBANDSR.

BANDPRIORITY a string of band names separated by commas, with no spaces, and not necessarily terminated with a comma. It gives a list of bands in priority order for selection for output by the control module BANDFILT.

BANDS a string of band names separated by commas, with no spaces, and not necessarily terminated with a comma. Tells the control module which bands to process. (In the case of INVERT, tells it which bands to look for asterisks in.) Default is null. If left null, the meaning depends on the control module, e.g. for BANDSORT it means select all bands (unless NOTBANDS is non-null), for BANDAID it means do nothing.

ENDEF a string used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph c.

FORCENKEY a string used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph d.

HANDLENAME In using TOAST control modules that sort on forms, or BIRDBATH modules that check alphabetical order, the user may supply a function to produce a "handle" for each form to be sorted. The default name for the function is 'HANDLE', but may in fact be any legitimate SNOBOL4 function name. If it is not 'HANDLE', the user must specify what it is through a SPECS statement e.g

```
HANDLENAME = 'FHANDLE'
```

This parameter is relevant to all TOAST control modules except REBANDLE, and to the BIRDBATH control module SEQCHECK. This parameter is given in the phase-1 SPECS file.

HANDLEBNAME Some TOAST control modules, e.g., INVERT, use a second handle function (see HANDLENAME, above) whose default name is 'HANDLEB'. If the user chooses to supply a function with another name, that name must be specified in a phase-1 SPECS statement, e.g.

```
HANDLEBNAME = 'ENGHAND'
```

In INVERT, this function is for sorting the keywords of the definition (see Appendix I, paragraph q.).

This parameter is relevant for the control modules INVERT and INVERT2, those that allow user-written handle functions to override the HANDLEB(X) default handle function. This parameter is given in the phase-1 SPECS file.

HEADHANG the number of character positions of hanging indent the headword bands will be printed with, in 1-format printed output. It is used by the LISTGEN() function. Default is 10.

This is one of the '1-format parameters', q.v. elsewhere in this section.

HOOK a string used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph g.

HWFIELD in BANDCORD and BANDSORT output, the width of the headword field (left hand column of the output). Any headword longer than this is truncated. Default is 35. It is given in the phase-1 SPECS file.

IDLEN the length of the fixed serial-number field (e.g. TSO line number) on input records. Used in the pattern R.INPAT, q.v. It is normally determined automatically by the first

call to READ(), but may be overridden. It is not usually relevant on the PC.

This is one of the 'Input parameters', q.v. elsewhere in this section.

Input parameters These are IDLEN, PARAPAT, R.INPAT, STOPAFTER, and STOPPAT. Only STOPAFTER needs normally to be used. They are relevant to all the control modules except NULLRUN. In TOAST control modules they are relevant only to phase-1.

KEEPANGLE a flag used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph k.

KEEPBANDLABEL old name for PRINTBANDLABELS, a flag used by the INVERT and INVERT2 control modules in phase-2. See Appendix I, paragraph p.

KEEPSEQ a flag used by the BANDSORT control module to determine whether instances of the same band are to be subsorted by their body or are to be kept in the original input sequence. To obtain the latter, set this parameter to a non-null value. The default value is null, giving the former result. It is used in phase-1 of BANDSORT.

KEYMARK a string used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph f.

LFIELD in BANDCORD output, this is the width of the left context field in the center-format concordance of the band bodies. It is defaulted to 12. It is a phase-1 parameter.

l-format parameters These are HEADHANG, SUBENTINDENT, SUBMODEINDENT, and SUPBANDLABEL, plus the page-formatting parameters NOPAGE, LINESPERPAGE, LMARGIN, PAGECOUNT, RMARGIN, SUPHEADING, SUPPINSP, and TITLE. They are all relevant to control modules that can produce l-format output with more than one band per entry. In TOAST control modules they are relevant to phase-2. The page-formatting parameters alone are relevant also to non-l-format printed output, such as BANDSORT output.

LINESIZE the maximum length of lines for hanging paragraph output in p-format. Used by the WRITEBAND function. Default is 70.

This is one of the 'p-format parameters', q.v. elsewhere in this section.

LINESPERPAGE the maximum number of lines that should be printed on a page of page-formatted output (e.g. l-format or BANDSORT output). Used by the PRINTL function. Default is 60.

This is a 'page-formatting parameter'. See under 'l-format

parameters' elsewhere in this section.

LISTGEN format parameters see '1-format parameters'.

LMARGIN position of left margin in print-formatted output. Used by the PRINTL function. Default is 0.

This is a 'page-formatting parameter'. See under '1-format parameters' elsewhere in this section.

NOBANDS a flag used by certain control modules that select bands in addition to certain fixed bands such as the headword band. If set to a non-null value, it causes no additional bands to be selected. It is mutually exclusive with ALLBANDS, BANDS, and NOTBANDS. It is used by the function IFBANDQ(). NOBANDS is relevant to the control modules BANDFILT and ONBANDSR.

NOPAGE When this flag is set to a non-null number, it causes page-skipping to be suppressed, along with printing of titles and page numbers. It is available in the PC version, not on the mainframe. It is useful if an output file is not to be printed but will only be examined on the screen. It is used by the PRINTL function. It renders the parameters LINESPERPAGE, TITLE, and SUPHEADING irrelevant.

This is a 'page-formatting parameter'. See under '1-format parameters' elsewhere in this section.

NONUM In p-format output a TSO-style line number (8 digits, increasing by 10's) is normally generated and attached to the front of every record written out. If this flag is set to a non-null value, no such number is attached. Used by the function NUMFIELD which is called by the function WRITE. Default in the mainframe version is null: the number field is generated. In the PC version the opposite is true.

This is a 'p-format parameter', q.v. elsewhere in this section.

NOTBANDS a string consisting of bandnames separated by commas, with no spaces, and not necessarily terminated by comma. Some control modules allow you to use this parameter instead of the parameter BANDS (q.v.), to specify what bands are NOT to be processed rather than what bands ARE to be processed. These control modules are BANDFILT, LISTGEN, ONBANDS, ONBANDSR, BANDSORT, and XCHECK. In TOAST control modules, (BANDSORT and XCHECK), this parameter is relevant to phase-1.

NOWRITE In the BANDAID control module, any band that is not mentioned in the BANDS parameter is normally written out. To prevent the unmentioned bands from being written out, set this parameter to a non-null value.

OFFSET in any outputting of hanging paragraphs (see 3.2, d), this is the number of blanks to be prefixed to continuation lines.

The default is 1, and should not normally be changed because programs that read the data in again assume this. However, for BANDSORT output the default is 50, causing continuation lines to be indented out of the way of the headword field, which is normally 35 character positions wide (see HWFIELD parameter).

This parameter should not normally be reset, except possibly in phase-2 of BANDSORT. In addition to BANDSORT, it is used by all control modules that produce p-format or l-format output.

ONBANDS a list of band names separated by commas, as in the parameter BANDS, used by the ONBANDS and ONBANDSR control modules to determine whether a given entry is to be selected. It is selected if the entry contains a band in this list and if the pattern ONPAT matches in the body of the band. If the list is null, nothing is selected. The default is null. However, the parameter REJECT, q.v., in ONBANDSR, when set to a non-null value reverses the selection criteria.

ONPAT a pattern used by the ONBANDS and ONBANDSR control modules. See ONBANDS parameter, above. Default is null, which matches in all cases.

OPTP a string used by the INVERT and INVERT2 control modules in phase-1. See Appendix I, paragraph m.

page-formatting parameters see 'l-format parameters'.

PAGECOUNT the current page number. The default initially is 0. To start printing with another number, set this parameter equal to one less than that number. Used by the PRINTL function.

This is a 'page-formatting parameter'. See under 'l-format parameters' elsewhere in this section.

PARAPAT the pattern used to parse each paragraph read in into its components according to band-format conventions: headword level (the dots, if any), mode numbers (if any), band name (obligatory), and body (if any). Used by the READ function in BIRDBATH and in phase-1 of TOAST, but not the READ function of TOAST phase-2. It does not need to be overridden unless the input does not follow standard band-format conventions. The default value may be examined in Program Segment 2.2.

This is one of the 'Input parameters', q.v. elsewhere in this section. It should never need to be overridden.

PATTERN a pattern used by the EDLISTB control module to determine whether a given band is to be selected for writing out. It is selected if the pattern matches in the body of the band. Default is null, which matches in all cases.

PFORMAT a flag that determines, in those control modules that allow this choice, whether output is to be in p-format or l-format. The former is suitable for output to a file that is to be further processed by these programs; the latter is for printed output. When not set (the default situation), the output is in l-format; when set to a non-null value the output is in p-format. This parameter is used by the function LISTORWRITE to determine whether to call LISTGEN (for l-format) or to call WRITEBAND (for p-format).

In TOAST control modules it is used in phase-2.

p-format parameters They are LINESIZE, NONUM, OFFSET, SUPPINSP. They are relevant to control modules that can produce p-format output. In TOAST control modules they are used in phase-2.

PHASE In TOAST runs, this number is normally automatically set at the beginning of a phase to either 1 or 2 depending on whether phase-1 or phase-2 is being executed. It normally would not have to be overridden.

PRINTBANDLABELS A flag used by the INVERT and INVERT2 control modules in phase-2. See Appendix I, paragraph p. The old name for this parameter was KEEPBANDLABEL.

R.INPAT the pattern used to separate the line number, if any, of each input record from the rest of the record. The default is TAB(IDLEN) . PG.ID2 REM . PG.BUF2
It is used by the function READPG.

It is one of the 'Input parameters', q.v. elsewhere in this section, but it should never need to be overridden.

REJECT a flag used by the ONBANDSR control module. It reverses the sense of the selection criteria specified by ONBANDS and ONPAT. That is, if ONBANDS and ONPAT are both satisfied, the band is rejected instead of accepted.

RMARGIN position of right margin in print-formatted output. Used by the PRINTL function. On the mainframe version the default is 132, which is the maximum width of wide printout paper. On the PC version it is 79.

This is one of the 'page-formatting parameters'. See under 'l-format parameters' elsewhere in this section.

SEPARATION in BANDSORT output, this is the number of blank lines to be inserted when the band name changes. The default is 1. If it is set to a number greater than LINESPERPAGE, e.g. 80, then every different band will begin on a new page. It is used in phase-2 of BANDSORT.

SNUM Some control modules that print paragraphs from a file, e.g. the LISTGEN control module, normally print the line

number field, if any, from the records of the input file. To suppress these numbers, set this parameter to a non-null value. The default is null. Used by the function SERFLD.

This parameter is currently relevant only to the LISTGEN control module, and then only when the input file has a line number field.

STOPAFTER a number used to limit the amount of input read by the READ function. After this many paragraphs the READ function will look for a paragraph satisfying the pattern STOPPAT, and stop before that paragraph, behaving as if the end of the file had been reached. If this parameter is null, no limit is imposed. The default is null.

This is one of the 'Input parameters', q.v. elsewhere in this section.

STOPPAT the pattern used in conjunction with the STOPAFTER parameter (above). Default is POS(0) '.' NOTANY('.') which matches the first band of an entry.

SUBENTINDENT in 1-format output, the number of character positions that each level of subentries is further indented from the level above. Used by the LISTGEN function. Default is 3.

This is one of the '1-format parameters', q.v. elsewhere in this section.

SUBMODEINDENT in 1-format output, the number of character positions that each mode level is further indented from the higher level. Used by the LISTGEN function. Default is 2.

This is one of the '1-format parameters', q.v. elsewhere in this section.

SUBSUB This phase-1 flag, when set to a non-null value, is used to inform the TOAST control module SORTSUB that sub-subentries and deeper entries are all to be re-ordered. Otherwise SORTSUB will only re-order subentries. See SORTSUB in 3.5 for details.

SUPBANDLABEL in 1-format output, this flag determines whether the band label is to be printed or suppressed. It is suppressed if this parameter is set to a non-null value. Default is null. Used by the LISTGEN function. (However, for INVERT and INVERT2 output, see description of the parameter PRINTBANDLABELS.)

This is one of the '1-format parameters', q.v. elsewhere in this section.

SUPDUPTRACE a flag which, if set to a non-null value in phase-2

of a HANDSORT run, suppresses tracing of duplicate handles. The default is null, which causes instances of duplicate handles to be printed in the messages file ("BIRDBATH RUN LOG") of phase-2 of the run.

SUPHEADING in print-formatted (e.g. 1-format or BANDSORT) output, this flag determines whether a running head, is to be printed or not. It would appear in the right hand corner under the page number. It is not printed if this parameter is set to a non-null value. Default is null (running head is printed). Used by the PRINTL function.

This is one of the 'page-formatting parameters'. See under '1-format parameters' elsewhere in this section.

SUPPINSP in outputting of hanging paragraphs, whether p-format or 1-format or any other formats, this flag determines whether spaces in the text that happen to fall at the beginning of continuation lines in the output paragraph are to be discarded or not. They are discarded if this parameter is set to a non-null value. Default is null. Used by the BREAKP function.

This is in both the families of 'p-format parameters' and '1-format parameters', q.v. elsewhere in this section. However, it should never need to be overridden.

TBANDS a string of one or more bandnames separated by commas, with no spaces, and not necessarily terminated by comma, used by the INVERT2 control module in phase-1. See Appendix I, paragraph r.

TITLE in print-formatted output, this is a string that is printed at the top center of each page. Used by the PRINTL function. Default is the null string. It is reset when a .TITLE band is encountered in the data being formatted (see 3.3.3).

This is one of the 'page-formatting parameters'. See under '1-format parameters' elsewhere in this section.

TRACEHANDLE a flag which, if set to a non-null value in phase-2 of TOAST runs, causes a trace of the sort handles to be listed in sorted order in the messages file (the "BIRDBATH RUN LOG") of phase-2. No regular output is generated. This is only for checking out handle functions. A typical use is to test a handle function by HANDSORTing a file consisting only of test headwords (perhaps constructed especially for the purpose) and nothing else (no other bands). With TRACEHANDLE set, no output is generated except the sorted list of headwords with their handles, which can be easily inspected for correctness of order and for the reason for any incorrect ordering.

TRUSSEL a flag used by the INVERT and INVERT2 control modules in phase-2. See Appendix I, paragraph p.

XBANDS a string of one or more bandnames separated by commas, with no spaces, and not necessarily terminated by comma, used by the INVERT2 control module in phase-1. See Appendix I, paragraph r.

XENKEY a string used by the INVERT and INVERT2 control modules. See Appendix I, paragraph d.

3.7 GUIDE TO CHOOSING CONTROL MODULES AND PARAMETERS

The headings in this section describe a few commonly needed operations such as making a working printout, or extracting a subset of the file, and each is followed by some advice about which control module(s) can help you meet these needs. Not all needs are anticipated, by any means. Although some advice and examples are given regarding the use of several control modules, the reader should not rely on this section, but rather on sections 3.4 and 3.5, for a more complete description of the capabilities of each of the control modules.

3.7.1 MAKING A WORKING PRINTOUT

A printout of the masterfile is useful for proofreading and checking, for jotting corrections and additions on, for sending to colleagues, for taking back to the field, etc. The file itself can be printed out as-is, of course, and no control module is needed, or available, to do this. Any printing command or utility program available on the computer may be used. However, being in "p-format" such a printout is not very readable in that it does not clearly show the structure within an entry. The l-format output produced by the LISTGEN control module is designed to do this. In the PC environment LISTGEN is normally used to produce an l-format file rather than a printout directly. Then the system print command can be used to print out that file.

LISTGEN can be used, of course, to produce l-format output from any p-format file, not only a masterfile. For instance, in those cases where it is necessary to request p-format output from a BIRDBATH or TOAST run because the output is to be further processed (used as input), an l-format version of the p-format output may be made with LISTGEN.

With LISTGEN, you should normally provide some informative title to be printed at the top of each page, using the SPECS parameter TITLE. The title should be short but give enough distinctive information about the file being printed, along with the date, so that you will be able to tell at a glance, possibly weeks or months later, what this printout was about. An untitled printout can quickly become meaningless. The right margin parameter RMARGIN is pre-set on the IBM mainframe version and on the PC version to agree with the typical paper widths used in the two environments: 132 on the mainframe and 79 on the PC. The left margin parameter LMARGIN is 0 in both cases. You may override these defaults with SPECS statements. Other parameters control the different indentations in the printout (see description of LISTGEN in 3.4), and at some point you may wish to experiment with overriding them. In the PC version there is another parameter, NOPAGE, a flag which allows you to suppress pagination and the accompanying periodic interruption of the printout with title lines and page numbers. You might want to

do this if the "printout" is actually not destined for printing out, but only for browsing on the screen.

3.7.2 PROOFREADING AND CHECKING FOR ERRORS AND INCONSISTENCIES

This category covers a great range of actual activities. For straight proofreading against, say, the field notebook from which the data were entered into the file, the Listgen output described above is the usual tool.

Remember that any corrections and editorial changes suggested by perusing these outputs must be made back in the masterfile and not in the output files. If the bandsort file were edited, for instance, the edits cannot automatically be posted back into the masterfile.

For proofing and checking that is not tied directly to a source, but done by scanning a printout or a file, any format that serves to focus the reader's attention on some limited aspect of the data can be very effective. A simple example of such a format is a selective Listgen, one that shows only a small number of different bands--say only the headword and definition bands, or only the headword and illustration bands. Such a printout, or file, can be made with the LISTGEN control module, with the bands to be selected being specified through the BANDS parameter. If the desired bands do not occur in all entries, then the control module ONBANDSR may be more appropriate because it does not select every headword as LISTGEN does. For ONBANDSR, the criterion bands are specified through the parameter ONBANDS.

Any rearrangement that tends to bring together similar features in the data is useful in making errors and inconsistencies easier to spot. The output of the BANDSORT control module illustrates this. This output brings together all instances of a given band and exhibits them in sorted order by the body of the band. While perusing such an output one can focus on the contents and conventions of a single band at a time, undistracted by other intervening bands. The bodies being in sorted order also produces some peculiar effects advantageous to the checking process. To make a bandsort of certain bands, specify what bands with the SPECS parameter BANDS, or exclude certain bands by using NOTBANDS. If neither is mentioned then a complete bandsort is made.

The band concordance, produced by the BANDCORD control module, has similar effects, although that output tends to be more voluminous, especially for bands that typically have long bodies. A band concordance would be more useful for bands that typically contain long strings of text (as in an illustration band) rather than single words or abbreviations (such as a part-of-speech band), in which case it would reduce to a bandsort. It would also probably be more useful if it is made of a single band or of a set of closely related bands (in the sense of containing similar types of

information) than if several heterogeneous bands were mixed together. Specify what band(s) to concord with the BANDS parameter.

The finderlist, produced by INVERT or INVERT2, usually brings out inconsistencies in wording, as well as typos and misspellings, in definition bands. Use the BANDS parameter to specify what bands to make the finderlist from.

Beyond this passive scanning of transformed files, it is possible to actively go after known types of errors or inconsistencies directly in the masterfile, with the help of various control modules. ONBANDSR, for instance, can be used to extract all entries that contain a given band with a certain pattern in it. The pattern is specified in ONPAT and the band is specified in the ONBANDS parameter (more than one band may be specified). You need, however, to know a little about patterns. If, for instance, only certain symbols (aiuptkh&#:) are allowed in a pronunciation band PR, then it would be possibly to ferret out all entries whose PR band contained any other characters by using these SPECS settings:

```
ONBANDS  =  'PR'
ONPAT    =  NOTANY('aiuptkh&#:')
```

Another example: To find all entries that do not contain certain bands, set ONBANDS to those bands, and set REJECT, e.g.:

```
ONBANDS = 'df,gl,des'
REJECT  = 1
```

Note that you cannot select exclusively for bands that fail to contain a given pattern by setting ONBANDS, ONPAT, and REJECT, as in

```
ONBANDS  =  'df,gl'
ONPAT    =  '*'
REJECT   =  1
```

If we label three classes of entries:
A. entries with df or gl bands which contain *
B. entries with df or gl bands which contain no *
C. entries with no df or gl bands
The above specs would select both B and C. Currently there is no simple setting of parameters that would select B alone.

Another type of checking that can be done is to search for misalphabetized entries in a masterfile, say after entries have been inserted by editing. This is done with the BIRDBATH module SEQCHECK, in conjunction with a specially written function called a handle which describes the alphabetical order. SEQCHECK notes all pairs of headwords that are not in strictly ascending alphabetical order according to this handle, but does not put them into the right order. More often than not, this exercise reveals not only items that are truly out of order, but also details in the alphabetical

order that have not been addressed. These details would not have been brought out if the file had simply been alphabetized without the benefit of a checking run. To use SEQCHECK requires some programming, and an understanding of handle-making in relation to alphabetical order, in order to construct the handle function. SEQCHECK requires one SPECS parameter, HANDLENAME, giving the name of the handle function. The function itself is placed in the USERFNS file.

A final example is checking for "unresolved cross-references". When a form is referred to but does not occur as a main headword in the file, then the referenced form is "unresolved"--it is pointing to nothing, leading the reader on a fruitless chase. The TOAST control module XCHECK locates these unresolved references. To use XCHECK, the SPECS parameter BANDS must be supplied, naming the band(s) which contain cross-reference forms. In addition, a user-written function XREF() must be supplied to extract the forms from these bands. The XCHECK control module itself should be examined for further comments on programming for it.

3.7.3 EXTRACTING A SUBSET OF THE FILE

One often needs to extract material from the file for special purposes: entries of ethnobotanical interest, astronomical terms, loan words, slang words, comparative data, etc. etc. If the material has been properly structured into bands, the extraction specifications can usually be stated simply in terms of bands.

The simplest case is to retrieve all entries that contain a specified band. ONBANDS2 can be used, and the band is specified with the BANDS parameter. More than one band can be listed in the BANDS parameter, in which case entries containing any of the bands will be selected. One can further specify that the body of the band should satisfy a given pattern, given as ONPAT, before the entry should be selected. For instance, assuming an appropriate band structure, these two SPECS statements might cause all English loanwords to be retrieved:

```
ONBANDS  =  'LOAN'
ONPAT    =  'Eng'
```

With ONBANDS2 the entire entry and all its subentries are retrieved, which may be more information than is desired. ONBANDSR, on the other hand, provides facilities for finer tuning of the what material is to be retrieved.

ONBANDSR has avilable, in addition to these two parameters, another set of parameters that determines what bands are retrieved. If no other parameters are specified, then all bands are retrieved (except for dominating entries and subentries which do not themselves satisfy the search criteria--and they are represtened

only by their headword bands). If NOBANDS is set to 1, then no other bands besides the headword band and the ONBANDS bands are retrieved. If the BANDS parameter is specified, then the bands given by the BANDS parameter are retrieved in addition. Thus is would be possible, for instance, to retrieve for ethnobotanical purposes only those bands of botanical interest.

If the subset conditions can be stated only in terms of bands, and not their contents, then the BIRDBATH control module BANDFILT might be appropriate. Although it provides a complicated set of parameters for specifying what bands are to be retrieved, the simplest parameter to use, BANDS, can be useful alone. Simply set BANDS equal to the bands to be retrieved. The program will retrieve all headwords, plus any occurrences of the specified bands in every entry.

The output from all these programs are normally in l-format. If it is necessary to run further programs on the output, as when a finderlist is to be generated from the ethnobotanical subfile, then the output must be in p-format. This is ensured by setting PFORMAT = 1.

3.7.4 ALPHABETIZING THE MASTERFILE

Note that none of the operations described above do any re-ordering of material in the masterfile. The masterfile is normally maintained in alphabetical order by headword, so that a listgen of it will be in alphabetical order, and entries can be easily found (or determined to be absent) for editing purposes in the file or in the printout. However, it may get out of alphabetical order for any of a number of reasons: entries may have been inserted in the wrong place, a batch of new entries may have been entered at the end of the file, the forms of certain headwords may have been edited and changed, or it may have been decided to change the rules of alphabetical order. Also, two or more files from different sources may need to be combined into a single alphabetized sequence.

Alphabetizing a masterfile is done with the TOAST control module HANDSORT. The "HAND" part of this name reflects the fact that a "handle" function needs to be written to describe the alphabetical order, and the user provides this in the USERFNS file. Since the sorted file will normally become the new masterfile, it should normally be in p-format, so that it can be further processed (l-format is primarily for human consumption and cannot be further processed). To cause the output to be in p-format, set PFORMAT = 1 in the phase-2 SPECS file of the HANDSORT run. HANDSORT output can be in l-format if it is not to be used for further processing, such as a HANDSORT on the output of the XCHECK module.

3.7.5 MAKING CLASSIFIED LISTS

Entries can be classified according to the information in a given band, such as a part-of-speech band or a semantic field band. A classification by part-of-speech, for instance, would place into separate groups all nouns, all verbs, etc. A semantic field classification would group together all kinship terms, all words for geographic features, and so forth. A bandsort provides a crude classified list of this kind. A bandsort of the part-of-speech band, for instance, since it is sorted by the body of the band, groups together all nouns, then all verbs, etc. The headwords appear at the left. Note that a headword will not appear in the bandsort if the relevant band does not occur in the entry for that headword. For a bandsort of just certain bands, use the phase-1 SPECS parameter BANDS to specify the bands.

The bandsort format is not designed specifically for this purpose--it serves many different uses. It happens to be useful as a classified list, but for this purpose the format is not the most elegant. For one thing, the bodies of the bands appear to be repetitive and redundant. (For a different kind of band, say definition bands, no two of which are identical, this will not be the case.) Another limitation is that only the headwords are given, and there is no way to display more of the entry. By contrast, the TOAST control module SORTENT gives a classified file of entries rather than a list of only headwords. It can also classify by any arbitrary material in the entry, not just the contents of a given band. However, to use it requires writing two functions to be inserted via the "user functions" file. The comments in the control module itself should be consulted for the requirements for writing these functions.

3.7.6 MAKING INDEXES AND OTHER APPARATUS

The bandsort, again, provides a crude kind of index to the headwords by the contents of given bands, in that all the occurrences of a given band are grouped together, sorted by the body of the band, with the headwords appearing at the left.

The finderlist, generated by the TOAST control module INVERT or INVERT2, also provides an index to headwords, but sorted by keywords occurring in certain bands, usually the definition band(s). Keywords are words flagged by the user with (normally) an asterisk. Although often considered as something approximating a reverse dictionary (e.g. English-to-target language), it is in fact little more than an index of keywords in the definitions. It is, however, a two-level index. The index at the back of this book is an example of a two-level index. (In fact there are a few three-level index entries.) There is an entry something like this for the term "bands", showing the second level by indentation:

```
bands,  23, 26
   headword,  24
   sequence of,  24, 31
   variety of,  13, 37
```

In an analogous way, a finderlist might have an entry for "basket" that looks something like this:

```
basket::  oruikl, sualo
    basket used for cooked food::  orekill
    fishing basket::  chelais, cheleuocho
    small woven basket::  tet
```

In the finderlist the first-level entries (e.g. "basket", in this example) are called "keywords", and the subordinate entries ("fishing basket", etc.) "phrases". These structures are identified in the definition bands by the program by means of special marks and ordinary punctuation symbols, as explained in Appendix I. The only SPECS parameter required is the BANDS parameter, specifying which band(s) to look for keywords in. Other parameters inform the program if you have used different characters from the normal ones for delimiting keyword and phrase.

The words after the double colon are, of course, headwords in the dictionary file. In the case of the headword of a subentry, conceivably it would be difficult for a user to find the subentry, since it would be attached to some main entry. In a finderlist made by INVERT2 however, the main entry headword is provided along with each subentry headword, in the format:

```
subheadword < mainheadword
```

Thus, in the example above, if orekill had been a subentry under oruikl then the second line of the example would read:

```
basket used for cooked food::  orekill < oruikl
```

INVERT2 can index not only to subentries but also to example phrases and sentences, if these are organized into bands in a certain way. Please see paragraph r of Appendix I for details.

CHAPTER 4. RUNNING THE PROGRAMS ON A PERSONAL COMPUTER UNDER DOS

Assuming that you are familiar with the mechanics of using the DOS operating system, you will need to know how to set up and execute BIRDBATH and TOAST runs in DOS.

4.1 RUNNING BIRDBATH AND TOAST UNDER DOS

This section shows conventions you must follow and the various options you have in setting up a BIRDBATH or TOAST run on a DOS machine. It includes a few instructions on general DOS conventions, but otherwise assumes that you are somewhat familiar with DOS and the running of batch jobs under DOS.

Recall that a BIRDBATH run uses a BIRDBATH control module, and a TOAST run uses a TOAST control module. Do not try to use a BIRDBATH control module in a TOAST run or vice-versa. The description of control modules in Chapter 3 shows which control modules are BIRDBATH and which are TOAST.

Your copy of LEXWARE for the PC may include one or more batch files for executing a BIRDBATH or TOAST run. In addition, users may themselves construct batch files which better suit their own convenience, and share them with each other. Although none of these batch files are required for setting up and running BIRDBATH or TOAST (you may type a series of DOS commands yourself if you wish), they do make the task more convenient by demanding less typing. They are not part of the system of programs (as described, for instance, in Chapter 3 or 6) and may be altered by the user with no effect on the system itself.

When one of these batch files is called, the commands in the file assemble and execute the run. Some of the batch files are more sophisticated than others, and can check for the presence of needed files and other facilities, but are more complicated to use.

This chapter will describe the use of two particular batch files for executing a BIRDBATH run: BIRDBATH.BAT and BIRDBAT2.BAT, and two batch files for executing a TOAST run: TOAST.BAT and TOAST2.BAT. The first of each pair takes 4 arguments after the name of the command, e.g.

```
BIRDBATH  LISTGEN  ARAWAK.LEX  ARAWAK.LST  LIST.SPC
```

while the second normally takes 3 arguments, e.g.

```
BIRDBAT2  LISTGEN  ARAWAK.LEX  ARAWAK.LST
```

In each case, the first argument is the name of the control module, in this case LISTGEN. The second and third arguments are the input and output files, respectively.

The last argument of the BIRDBATH command is the name of the SPECS file to be used.

BIRDBAT2 does not have this argument. Instead, BIRDBAT2 assumes a SPECS file having the same name as the control module but with the extension .SPC. Thus in the above case, the SPECS file LISTGEN.SPC will be used.

TOAST and TOAST2 behave correspondingly.

All the commands assume that the programs are in a directory C:\LEXWARE. This can, of course, be changed by changing all references to C:\LEXWARE within the batch files. The second command of each pair, namely BIRDBAT2 and TOAST2 permit the user to refer to another directory without changing anything within the batch files. Instead, an environment variable LEXPATH is set to the new directory by using the SET command in DOS, e.g.

```
SET LEXPATH=D:\LEXWARE
```

or

```
SET LEXPATH=C:\LEXWARE2
```

Thus a new version of the programs can be kept in a separate directory and used without removing the old version.

---

a. THE SIMPLEST LISTGEN RUN

A LISTGEN run illustrates the simplest BIRDBATH run. It would be called exactly as in the examples given already:

```
BIRDBATH  LISTGEN  TEST.LEX  TEST.LST

or

BIRDBAT2  LISTGEN  TEST.LEX  TEST.LST
```

Figure 4.1.1

The name of the control module in this run is LISTGEN, and the name of the dictionary file being processed is TEST.LEX. The output is to go into the file TEST.LST. Of course, you should use the names of the files appropriate to your own purposes. No SPECS file has been specified in either case. If you use BIRDBATH, this will cause no SPECS file to be used. If you use BIRDBAT2, this will cause a file LISTGEN.SPC to be used as a SPECS file if it exists in the current directory; otherwise no SPECS file will be used.

If the run is successful, the file in Listgen format will be saved in TEST.LST, where it may be examined, and from which it may be printed using, for instance, the PRINT command in DOS.

b. SPECS file

The above run will generate a very basic Listgen file. One of the simplest ways to embelish it is to provide a title to be printed at the top of each page when this file is printed out. This can be done by using a SPECS file, and putting in it a statement that assigns a string to the variable TITLE:

```
TITLE = 'LISTGEN of TEST.LEX'
```

Note that there must be spaces (at least one space) at the left of the statement, and on either side of the = sign.

Another Listgen variable that can be assigned in the SPECS file is NOPAGE. Set NOPAGE equal to some non-null value (any value--often 1 is used.) to suppress pagination of the output. If the output file is to be perused on the screen only and not to be printed, this option is useful because it will produce a file that is not interrupted by page breaks and titles. The statement would be as follows:

```
NOPAGE = 1
```

Other SPECS parameters relevant to LISTGEN, and to any other control module that produces l-format output, are described in Section 3.4 under LISTGEN.

c. LIMITING THE AMOUNT OF INPUT

A particular SPECS parameter that is available for all runs is called STOPAFTER. This parameter may be set to some positive integer, as with this SPECS statement:

```
STOPAFTER  =  600
```

The program will then stop processing the input file after reading this many paragraphs. It will read to the end of the dictionary entry. It will not stop in the middle of an entry. The program actually looks for a pattern, STOPPAT, in the following paragraphs, and STOPPAT is initialized to match the first band of a main entry (POS(0) '.' NOTANY('.')). STOPPAT may be overridden in the SPECS file. (See Section 6.2.2 for further discussion of STOPAFTER and STOPPAT.) This parameter is useful for making limited trial runs.

d. SPECIAL CONTROL MODULES

If you are not using a control module from the library but are using one of your own, it should be placed in the directory in which the batch file is called, and you should use BIRDBAT2 or TOAST2. It

can be executed exactly as a regular control module, e.g. (if PEEL.SPT is the special control module),

```
BIRDBAT2  PEEL  X.LEX   X.OUT
```

A SPECS file (in this case PEEL.SPC) will be used. Actually, BIRDBAT2 and TOAST2 will look for the control module first in the current directory and then in the Lexware directory. However, it is recommended to keep the Lexware directory free of special-purpose control modules and other files. To call special control modules with the older batch files BIRDBATH and TOAST, however, the control modules must be in the Lexware directory.

To write your own control module you would of course have to know something about how BIRDBATH works and the facilities that it makes available (see Section 6.3).

e. CONCATENATING FILES FOR INPUT

If the dictionary file is contained in several physical files, they may be processed as a single file without concatenating them into a single physical file. Instead, a file containing the names of the individual files is constructed in band format, with special bandnames, and that file is used as input. (See Section 3.3.3, Uses of bands.) Thus if a "master file", say ARAWAK.MAS, as shown in Figure 4.1.2, is used as input, e.g. in the command

```
BIRDBAT2  LISTGEN  ARAWAK.MAS  ARAWAK.LST
```

```
.LIST    Arawak  file list
FILE     ARA.LEX
FILE     ARB.LEX
FILE     ARC.LEX
```

Figure 4.1.2, the file ARAWAK.MAS

then the three files ARA.LEX, ARB.LEX, and ARC.LEX will be treated as a single file and used for the actual input.

f. USER SUPPLIED FUNCTIONS

If you need to supply special-purpose functions, say a handle-making function, they may be placed in a file, and the command BIRDBAT2 or TOAST2 must be used. The name of this file must be given as the fourth argument to the command, e.g. PAQHAND.SPT contains user-supplied functions in these commands:

```
BIRDBAT2  SEQCHECK  PAQ.LEX  PAQ.CHK  PAQHAND.SPT

TOAST2    HANDSORT  PAQ.LEX  PAQ.SRT  PAQHAND.SPT
```

The old BIRDBATH and TOAST batch files do not provide this capability.

CHAPTER 5. TOPICS IN DICTIONARY PROCESSING

This chapter contains essays on various questions of strategy that frequently arise in processing dictionaries but which are not related directly or exclusively to any single program or control module discussed in the preceding chapter.

5.1 STORING THE DATA

The usual media for storing data for computer processing are punched cards, disk, and tape.

5.1.1 PUNCHED CARDS

Although punched cards continue to serve a useful function in running jobs at some computing centers, e.g. for submitting JOB control cards, they have been superseded as a medium for storing large amounts of data. Computer-oriented lexicographers have moved to tapes and disk files for a number of reasons. They are more flexible in terms of the choice of orthographic symbols available, and they are less cumbersome. Furthermore, disk files are easier to edit. Also, the cost of cards has risen dramatically in recent years. Since cards are not generally used as a means of storing data, they will not be discussed in any further detail in this manual.

5.1.2 DISK DATASETS

Currently, most data is stored in disk files for computer manipulation. Compared with cards and tape, disk storage is more flexible and convenient for both people and computers. It is also faster for a computer to access. It is, however, the most expensive of the three media. At most computer installations, disk files are subject to various restrictions on usage. The restrictions generally have to do with the amount of data that can be stored and the length of time a file can be stored. For instance, each user may be allotted a quota of disk space for all his/her files. Size of files may be limited by practical, economic, or hardware considerations. Also, files that have not been used for a certain length of time may be subject to automatic "archiving" (removal to a cheaper storage medium, namely tape), rendering them not instantaneously retrievable. Restrictions such as these are designed to encourage users to make more economical use of a relatively expensive resource. At most installations, such restrictions do not present a deterrent to disk usage.

Some of the restrictions on disk usage at one installation are discussed in Appendix II, Section 9.

5.1.3 TAPE

Tapes are the cheapest of the three media for storing large quantities of data, such as those encountered in dictionary work, and are for practical purposes not subject to the restrictions that affect disk storage. Because the computer operators must mount and dismount tapes, however, most installations favor disk jobs over tape jobs. They may charge for mounting tapes, charge more for reading/writing of tapes, give tape jobs lower priority, or restrict the type of tape jobs to, say, the copying of files between tape and disk. Nevertheless, tapes are useful, indeed necessary, for storage of inactive files, for archival and back-up storage, and for exchange of data between different installations.

Details of using tape at the University of Hawaii are discussed in Appendix II, Section 8. Using tape in a BIRDBATH run is discussed in Section 4.1.

5.2 PUNCTUATION AND TYPOGRAPHICAL CONSIDERATIONS

5.2.1 INTRODUCTION

If you decide to compile, or are compiling, a dictionary by means of a computer, you may or may not wish to pay attention to details that would ultimately affect the appearance of the dictionary as a typeset book.

On one hand, you may merely wish to utilize the computer to record large amounts of data, then sort, retrieve, and print out portions of that information in some rough form. In this case, you may not, in fact, intend to publish your data or dictionary at this point (or at all). If you are using the computer for data processing and retrieval purposes only, you may not wish to be encumbered with all of the details given in this section.

On the other hand, if you intend to have your dictionary published, you will want to have the material fully polished and encoded with the proper typesetting marks for sending to the printer. Since the dictionary is typeset directly from a computer medium, you will sooner or later have to pay attention to the minute details of punctuation, typography and style. How much attention you devote to these matters, and when, will depend on your own inclination, schedule, and how you spread your different tasks over the available time.

Some of your attention will be paid to matters of style, such as where to use italics, what to capitalize, what to put in parentheses, etc. These are mainly your own decisions. Nevertheless, this section includes a small subsection, 5.2.3, listing the most commonly occurring questions. Once you have decided on these matters, you will need ways to indicate change of font, etc., at particular places in your material. The bulk of this section, Section 5.2.2, presents conventions that have been developed during the processing of a number of dictionaries in band organization. These conventions will make it possible, or in some cases merely easier, for the user to indicate typographical functions.

When we speak of conventions, we mean either rules to follow, or processes that the computer (or, more properly, the system of programs) performs on your data. Sometimes the word "conventions" is used in both senses, as the senses are closely interrelated.

5.2.2 MECHANICAL CONSIDERATIONS

1. Punctuation

Unless otherwise indicated below, use punctuation marks as you would normally. One optional, non-ordinary convention that you may

wish to adopt is the following: you may consistently omit final punctuation in all occurrences of a given band. In the final copy, you can specify that the program is to append that particular punctuation to each occurrence of the band. This specification can be made on a band-by-band basis. Any regular, predictable, punctuation may be omitted from a band and inserted by a later program.

Another task you may wish to relegate to a program concerns placing round or square brackets around all the material in a specified band, relieving you of typing in this redundant material.

2. Capitalization

Normal capitalization practice can be followed, with a few exceptions. Headwords should not be capitalized unless they are proper names. Capitalizing all headwords indiscriminately "loses" this information. The beginning of definitions should not be capitalized unless the first word is a proper name. If desired, the first letter can be capitalized by a program later.

3. Typeface

You may specify what typeface you want material to be set in for publication by prefixing the material with the appropriate face-shift codes. The shift remains in effect until the end of the band or until it is countermanded by another intervening face-shift. For each band, a default type face may be specified, which will be the type face for that band when no instructions to the contrary are given. This default face will also apply if the band does not start with an explicit shift code. For most bands, this convention saves the trouble of putting in any shift code at all.

Five symbols are normally reserved for signalling changes in typeface in the eventual computer typesetting, though you may choose other symbols if you need these for representing other things. They are: the percent sign, %, the vertical bar, |, the dollar sign, $, the ampersand, &, and the 'at' sign, @. Their specific functions are described below.

These five symbols will not appear in the final copy. During the process of typesetting, the symbols will be deleted and the spaces they occupy closed up. Be aware of possible consequences of this when you are inserting these symbols. For example, if a word is to have a subscript on it, do not leave a space between the word and the dollar sign, $, that introduces the subscript: word$2 .

The symbols listed below are the normal shift codes:

Symbol	Shift Function
%	shift to italics
|	shift to roman (medium) face
$	shift to subscript (or superscript) (applies to next character only)
&	shift to boldface
@	shift to small caps

An alternative use of the @-sign (or any other symbol) is as an 'escape' character which can be used to create, with a specified following character, any other shift the user may desire and that the printing-house can provide. Using a double shift symbol is also a way of defining the shift codes when the normal code symbols are needed for orthographic purposes. For example, if & is needed as a text character, one might use @& as the boldface shift.

5.2.3 STYLISTIC CONSIDERATIONS

The preceding has been a description of mechanical conventions for indicating typographical functions which you may need but which cannot be directly represented on most common computer-processable media. This section suggests, among other things, some of the ways in which you may wish to use these typographical functions in a dictionary, i.e., when to use what face, where to leave spaces, etc. The computer is, of course, indifferent to whether you follow these suggestions. The conventions only have to do with the typographical appearance of your dictionary. You should also consult published dictionaries, other dictionaries now being processed in the computer and their authors, your publisher, lexicographical manuals (e.g., Zgusta 1971), and general style manuals (e.g., Univ. of Chicago 1949) before deciding on your own style. (See Section 5.2.4, References Cited.)

1. Subscripts on headwords.

If you have homophones (different words spelled in the same way), you may wish to disambiguate them orthographically in citations in your dictionary by using numerical subscripts (or superscripts). One consequence of not disambiguating homophones is that in the finderlist, the references to words that belong to homophonous sets will not be specific. A disadvantage to using subscripts is that, once you label a word with a subscript, you may feel it necessary to so label all other citations of the same word in any cross-references (not in running text). The references may be in synonym bands or "see also" lists within the entries. Currently, there is no mechanical aid to help you in finding all such occurrences of citations in the dictionary.

2. Latin abbreviations

These should not be italicized (except for "sic") and there should be no space within the abbreviations. Note: e.g., i.e., q.v.

3. Punctuation marks

The typeface of punctuation marks should be the same as that of the immediately preceding word unless the mark is the closing one of a pair of marks (parentheses, quotations). In this case the closing mark should agree in face with the opening one of the pair.

In most type fonts, quotation marks (both single and double) come in two varieties: opening and closing. On computer input and output (printout) fonts, there is no distinction. As on most typewriters, opening and closing quotes are symmetrical (" or '). When quotation marks are always used in pairs within a band, the programs that prepare the tape for the typesetter can easily distinguish intended opening from closing occurrences. Therefore, programs can be written to convert them to codes that would cause the appropriate left or right quotation mark to be typeset. A difficulty occurs when either quote mark (" or ') is used for other purposes, in which case they are not paired. This may occur in orthographies where " is used for rounding, or ' for glottal stop or as an apostrophe. A number of ways can be found around the problem, but they will not be discussed here. Probably the easiest solution is to request that all quotation marks be set as symmetrical marks. This solution may require going to another font for those particular characters.

4. Faces for categories of information

Headwords are usually set in boldface type. Citations of words within an entry, such as "see X, Y, and Z" are also normally in boldface.

Grammatical codes, base forms, etc., may be printed in small caps.

Occurrences of source-language words in running English text should be in italics, as should Latin scientific names (such as Linnaean binomials). Generally, in English definitions, which are in Roman face, any non-English words should be in italic type.

5.2.4 REFERENCES CITED

University of Chicago Press. 1949. A Manual of style: containing typographical and other rules for authors, printers, and publishers recommended by the University of Chicago Press. Chicago: University of Chicago Press.

Zgusta, Ladislav. 1971. Manual of Lexicography. The Hague: Mouton.

5.3 GUIDELINES FOR WRITING CORRECTIONS AND ADDITIONS ON PRINTOUTS

This section gives a number of guidelines for writing changes and additions on printouts of dictionaries. If someone besides the author will be entering the corrections into the computer, they need to be clearly marked on the printout.

1. Use a colored pencil or soft lead pencil.

2. Indicate reordering by means of arrows.

3. Text to be altered should not be obliterated. Mark through such material with a pencil and write the desired change nearby.

4. Text to be deleted should be circled (not obliterated), with a delete mark attached. Make clear whether surrounding blank spaces are also to be deleted. Spaces occupied by material to be deleted will be closed up unless the specific instruction is given that spaces be inserted.

5. Small amounts of text to be inserted may be written into the text directly. Use carets or arrows to indicate the precise point of insertion.

6. Larger blocks of text to be inserted may be written into the right margin or where there is sufficient blank space. Indicate explicitly by means of arrows where in the text such material is to be inserted. If necessary, continue on extra sheets, which should be attached to the page where the insertion is to be made.

7. Periods, commas, and other inconspicuous characters to be inserted should be emphasized by means of a small arrow, or as described in (8) below.

8. If the average number of changes and insertions to be made is less than one per page, a prominent mark should be made in the extreme right margin, level with each change or insertion to be made.

5.4 SORTING THE DICTIONARY

You may want to re-order the entries in a dictionary for a number of reasons. In one or more edits, you may have inserted entries in the wrong alphabetical position, or you may have added a batch of new entries at the end of the dictionary. You may want to sort together two or more dictionaries of the same language. Another reason for reordering is that you may have changed the spelling system and respelled all the headwords so that they are no longer in alphabetical order. Finally, you may want the order of entries to conform to a new set of ordering conventions. For any of these or other reasons you may wish to have the entries re-alphabetized.

Re-ordering is normally done by using TOAST with the HANDSORT control module to generate a re-alphabetized new master file. In using HANDSORT, the crucial information the user has to supply is what the desired alphabetical order is. If this information is not given, the program will assume a standard order (roughly that used in English dictionaries) in which hyphens and capitalizations are ignored. If you wish to specify a non-standard order, the information must be supplied in the form of a small program. This program is a function written in SPITBOL and inserted in the main program as Segment 6, USERFNS. This function should systematically generate from each headword a new pseudo-headword, called a "handle", which will be used in the actual sorting operation. Handles are thus headwords which are respelled in such a way that when they are sorted by the computer's standard alphabetical ("collating") order, the original headwords fall into the desired non-standard order.

If the handle function generates identical handles for two of the words in the dictionary the program will automatically preserve the original relative order of the words and print a message citing those two words. Identical handles could occur either because the two words are spelled identically to begin with or because the handle function obliterates the distinction between them. The messages allow you to determine easily which reason is the correct one in each case. If an underspecified handle is the cause, you may wish to refine it so that on a future sort the desired order will be obtained. You may want to deliberately underspecify the handle just so that the program will show you the conflicting words. Many errors can be located in this way. How this can happen will become apparent below.

If the handle function is at all complicated, it should be tested with some hand picked or constructed crucial example words before the entire dictionary is sorted. This testing can be done by making a set of entries with only headword bands containing the chosen examples. If you see anything wrong with the order of the sorted output, you can adjust the handle function and run the test again.

The HANDSORT control module sorts entries only according to the first band in each entry, the band normally used for the headword of the entry.

All subentries and sub-subentries, etc., under a given entry are carried along with their main entry in the sorting with no change of relative order among them.

An additional feature which will be put into the HANDSORT control module allows the sequence of the input dictionary file to be checked before sorting. Furthermore, sorting may be suppressed entirely, so that the run is made only for the purpose of sequence checking the dictionary. With the sequence check feature, the program will print out any pair of consecutive words in the input file that are not in increasing alphabetical order, that is, whose handles are not strictly increasing. Such a check can give an indication of whether a file needs sorting. It can also provide help in refining a proposed handle-making function when the input file is already largely in correct alphabetical order.

To refine a proposed handle, it is most instructive to start with a deliberately underspecified handle and to choose the direction in which to constrain it further by examining the words printed out in the sequence check messages. If one begins with a highly constrained and complicated handle, the effects on the order in the rarer cases will not be evident. A detailed examination of the entire list of sorted entries to find the rare occurrences would have to be made. Some effects of a complicated handle may be hard to anticipate, and may turn out to be unacceptable. With an underspecified handle, on the other hand, the program can help show in which directions the handle needs to be constrained with reference to the set of words in the given dictionary.

5.5 GENERATING INDEXES AND FINDERLISTS

Special conventions have been developed so that an index can be mechanically generated from specially marked 'keywords' in certain bands (See Appendix I "Finderlist Conventions.") These bands could be the regular definition and grammatical bands, for instance. With appropriate wording and use of ordinary or special punctuation in these bands, the index generated from the keywords can be useful as a finderlist in the target language (English for our purposes), even though it would not be a full-fledged dictionary of English words. The purpose of these conventions is to minimize the work necessary to compile the finderlist.

Not all of the functions necessary for making a finderlist can be performed by the computer automatically, however. Sometimes, even with these conventions, it would take a very unnatural wording of a definition in the dictionary to produce a natural sounding entry in the finderlist. For such situations, you may create and use a special band, which does not appear in the final published dictionary, in which you may construct phrases solely for the purpose of being copied into the finderlist. The "public" bands are then unencumbered by the necessity of being readable in two contexts simultaneously. These special "extract" bands would be subject to the same conventions as the public bands. The finderlist program can extract keywords from any number of specified bands.

Finally, if in a few cases even the above method fails to yield appropriately phrased finderlist entries, you can resort to editing of the finderlist.

The finderlist program need not be used to generate only finderlists. Another common use is to generate a classified index of terms for semantic domains, such as 'house parts', 'kinship terms', 'geographical terms', etc. What is needed is to do this is a special band (call it a semantics band) for each entry, containing the phrase 'house parts', etc. marked with the same conventions. The finderlist program is then told to extract from this band instead of the definition band.

Another kind of index, simpler in form than a finderlist, can be obtained from the BANDSORT program. (See Chapter 2 for a simple example.) No special marking within bands is necessary. The program takes all the bands and sorts them by band name and body, giving a simple index to the headwords based on band name and body.

More specialized indexes can be made by means of specially written control modules for TOAST.

CHAPTER 6. A TECHNICAL DESCRIPTION OF THE SYSTEM

This chapter is an overview and description, for the programmer, of the BIRDBATH and TOAST systems of programs. Such information is necessary if you are either maintaining the programs or first installing the systems on a computer. Some of this information is necessary if you need to write new control modules or complicated SPECS statements.

It is assumed in this chapter that you have read Chapters 3 and 4 of this manual and that you are familiar with SNOBOL4 and SPITBOL.

6.1 STRUCTURE OF THE SYSTEM

BIRDBATH is a collection of program files written in the SNOBOL4 programming language. A program would normally be run using one of the fast compilers, such as SPITBOL, SITBOL, or Macro SPITBOL. However, running a program under the much slower original interpretive Macro Implementation of SNOBOL4 is by no means precluded. Compilation is extremely fast, accounting for a negligible proportion of the cost of any but the shortest runs. Also, calling of pre-compiled functions is either not possible or awkward and implementation- and environment-dependent. For of these reasons each run starts by compiling the entire program. Modularity and flexibility are achieved by organizing the program as a sequence of source files (called a "sandwich") in which certain slots can be filled by interchangeable modules. All operating systems apparently have convenient ways of concatenating source files at compilation time. Therefore, the sandwich organization is easily implementable on all systems that have some SNOBOL4 compiler. The source program files can thus be maintained independently of the implementation and operating environment. Only the method of concatenating the source files needs to be tailored for each operating system. Further ways in which the programs have been shielded from differences in the compilers will be mentioned at the appropriate places.

The BIRDBATH "sandwich" consists of several "layers", each filled by a fixed or variable file of source language (SNOBOL4) statements. Each layer consists of one or more "segments" each of which consists of statements and functions oriented to particular purposes. The largest portion of the program system is devoted to I/O, as will be seen. Layers are separate physical files, while the "segments" are more or less logical divisions.

Layer			Segment	
1.	INIT		1.1	Initializations
			1.2	Functions needed for Macro-SNOBOL4 implementations
2.	SPECS	*	1.3	User-supplied parameter-assignment statements
3.	CORE		2.	General input functions
			3.	General output functions
			4.	Band processing functions, divided among various segments
(4.	SORTIO		5.	Functions for managing I/O for sorting, for TOAST)
5.	USERFNS	*	6.	Any special-purpose user-supplied functions
6.	CONTROL	*	7.	The control module

Ignoring Layer 4 for now, Layers 1-6 make up the source language program file for a BIRDBATH run. Layers 1-5 contain initializations and functions, while Layer 6 (Segment 7) contains the executive control structure that calls on various previously defined functions. It is the control module of Layer 6 that causes anything to happen; all the previous segments only set up parameters, tables, functions, etc. The layers marked with asterisks (*) are supplied by the user for a particular run. The other layers are fixed as far as the user is concerned. The control module is supplied by the user either by choosing from among an existing library of modules or by writing one or having one written.

How these five or six physically separate source files are actually brought together into one program depends on how this system is installed on a particular computer system.

A BIRDBATH program consists of Segments 1-7, without Segment 5. (Segment 6 is optional.) It involves no sorting. A TOAST run, on the other hand, consists of two BIRDBATH programs (each containing Segment 5) with a sort in between (Segment 6 is again optional):

```
TOAST Run:  phase 1 BIRDBATH (including SEGMENT 5)
            SORT
            phase 2 BIRDBATH (including SEGMENT 5)
```

A TOAST run uses the standard sort program on whatever computer system the programs are being run on. The first phase generates records to be sorted, the SORT sorts them, the second phase reads the sorted records and formats them as necessary for output. Segment 5 of the BIRDBATH program, SORTIO, manages both the output formating of the sort records before sorting and the reading in and deformating of the sorted records after sorting. In order to simplify program maintenance, the system was designed so that the source programs for both phases are physically identical. The possible exceptions are the user-supplied SPECS and USERFNS layers. The same CORE functions are available in both phases, and the same

SORTIO and the same control module appear in both phases of a given TOAST run. Of course, a different part of the control module is actually executed depending on which phase the program finds itself in. A small function in SORTIO called DETPHASE() determines which phase it is in.

Remember that how the three steps of a TOAST run are assembled and how the intercommunication between adjacent steps is accomplished are specific to a particular installation.

The next section, 6.2, describes the functions and other facilities provided in the fixed portions of the BIRDBATH program, segment by segment. The section after that, 6.3, gives some hints on how to write new control modules.

6.2 BUILT-IN FUNCTIONS AND OTHER FACILITIES

Please obtain and refer to a program listing while reading this section. Comments in the program will generally not be repeated here. A program listing may be obtained from any BIRDBATH or TOAST run. In particular, a 'null run' may be used to get such a listing. A null run of TOAST will result in a listing that includes Segment 5. A null run is one that uses the control module NULLRUN.

6.2.1 PROGRAM SEGMENT 1

6.2.1.1 PROGRAM SEGMENT 1.1: INITIALIZATIONS and INTERFACING

Certain SNOBOL4 keywords are first set by the program:

```
&STLIMIT = 10000000
```

Allows processing of large files without being aborted by the statement counter. Note that, under MACRO SPITBOL on the PDP-11 family of computers, there is a limit of 32K on integers.

```
&FULLSCAN = 1
```

Allows greater efficiency of pattern matching under SPITBOL. However, QUICKSCAN mode is faster in MACRO SNOBOL4. It is believed that no built-in patterns in these programs depend on FULLSCAN to operate properly.

```
&DUMP = 2
```

Produces a complete dump of natural variables and tables and arrays at termination.

The next statements set up variables for use in the upper and lower case translation functions. These functions, DESHIFT(X) and UPSHIFT(X), return respectively the lower-case and upper-case translation of their arguments. The unary operators '/' and '#' are OPSYNed by the program to DESHIFT and UPSHIFT respectively, to allow easy application of these functions.

Next come INPUT and OUTPUT associations, and the lowest level output functions. Outputing throughout the system is generally done only through calls to the first four of the following functions:

WRITE(L)	meant for writing out a single line, with optionally a prefixed TSO-style line number generated by NUMFIELD() (see below).
PRINT(L)	writes out a single line, with a blank prefixed, for those printing systems that recognize this as a carriage-control character for single-space.

PRINTPAGE(L) writes out a single line, with a '1' prefixed, for those printing systems that recognize this as a carriage-control character for skipping a page.

MSG(SRC,TEXT) writes out a "source" and a message text to the MESSAGE file, the BIRDBATH Run Log.

NUMFIELD() This function generates an 8-digit TSO-style line number. Each time it is called it returns the next number, with an increment of 10. This is used only by WRITE(L). It may be suppressed by setting NONUM = 1 in the SPECS file.

STOP() (known also as TERMINATE() in an older version.) The calling of this function by the control module is the normal way to terminate the program. It prints out termination messages and transfers to END. Also, when it detects that it is in the first phase of a TOAST run, it writes out the sort control record. It then also sets &CODE to 1, which in the JCL procedure set up on the IBM 370 at the UHCC signals that the following step, the sort, can proceed. These interfacing details may have to be changed, or may not be relevant, for other installations.

a. Notes on input/output

Because of the variability in the I/O interface among different SNOBOL4 and SPITBOL implementations and the consequent likelihood that the low-level I/O operations will have to be adjusted for different installations, an effort has been made to confine such operations to a limited number of locations in the program. SEGMENT 1.1 is the main place where output happens. SEGMENT 2 is the only place where input happens, through the READ() function. In addition, in SEGMENT 5 (SORTIO) is a small function, WRITESORTCNTL(), which writes the sort control statement to a file SORTCNTL (in the IBM 360 SORT format).

b. Another note on I/O.

The program deals with only one input file and one output file for data. Another output file, MESSAGE, is for the run log. For sort jobs there is the very short, one-record SORTCNTL file. These are the only files the built-in facilities use. The user may, of course, program for additional files.

6.2.1.2 PROGRAM SEGMENT 1.2: HANDLE-MAKING AND TRANSLITERATION UTILITIES

These are four functions which can be handy in writing handle-making and transliteration functions. They are placed here so that the user may call them as early in the program as the SPECS file. These functions are useful for those components of handle-making and transliteration which require simple context-free substring-to-substring mapping. The functions allow the easy construction of tables of these mappings and then the actual mapping of strings using these tables. They are used as follows.

The function OPENTABLE('name',size) must be called first. It sets up a mapping table with the given name, and "opens" it (makes it available) for filling. The function works in conjunction with the next function, FILLTABLE, q.v. Note that the name of the table must be given in quotes in the function call, e.g. OPENTABLE('LUSHMAP',50) . Note also that the name must not conflict with any other identifier in the entire program, in particular with the name of the function that uses these utilities. The second argument may be left out; it is an estimate of the size of the table.

When a table is open, mappings, in the form of pairs of argument and value, can be put into it by calls to the function FILLTABLE(arg,val). The otherwise unused binary operator # has been OPSYNed to FILLTABLE so that calls to FILLTABLE may be simply expressed as

```
arg # val
```

Thus, loading a table consists in a call to OPENTABLE followed by a series of binary # expressions, e.g.

```
OPENTABLE('SKT')
  'k'  #  '01'
  'kh' #  '02'
  'g'  #  '03'
  'gh' #  '04'
       .
       .
       .
```

The order of input to the table is not relevant, except that should there be a duplicate argument, the value of the latter overrides. Values, of course, need not be unique. Values must not, however, be the null string. In fact, removing an entry from a table means to replace its value by the null string:

```
'y'  #  ''
```

Also, arguments cannot be the null string. This statement would lead to a loop, and such a statement is not checked for by the program:

```
'' # 'x'
```

Any number of such tables can be set up, but only one table can be open at a time. When a call to OPENTABLE is made, whatever table is currently open is closed and the new one opened. A table can be re-opened simply by calling OPENTABLE again with the name of that table (the table is not deleted; it is re-opened so that entries can be added to it). Calls to # load entries to (or change entries in) whatever table is open at the time of the call.

After a table is made, it can be used in two mapping functions, HANDLEMAP(s,'name') and TRANSMAP(s,'name'), where 'name' must be the name of a table previously established by the above functions. The name must be in quotes, e.g. HANDLEMAP(WORD,'LUSHMAP'). The first argument is the string to be mapped by the table. The value of the function is the result of the substring-by-substring mapping. Both functions operate in the same way, the only difference being in what they do with substrings not found as arguments in the table.

Both functions scan the string to be mapped from left to right, and at each position find the longest substring starting at that point for which there is a non-null argument in the table (whose name is given as the second argument of the function call). After replacing that substring with the corresponding value from the table, scanning resumes after the end of the replacement. If no argument can be found for a subtring of any length at a given position, then the character at that position is either skipped (ignored) or taken as its own mapping, depending on which function is being used. HANDLEMAP skips the character at that position and continues scanning after that position. TRANSMAP, on the other hand, retains that character and continues scanning after that position. This difference is motivated by the common need, in making handles, for segments to be "ignored", while in transliterations mappings frequently are sparsely specified and unmentioned segments are assumed to be unchanged and retained. However, both functions collect one copy of each different orphan character encountered while mapping with a given table, and strings them into a special string whose name is composed of 'UNRECSEG.' concatenated with the name of the table, e.g., UNRECSEG.LUSHMAP . The contents of this string may be examined by the programmer in the Dump of Natural Variables.

6.2.1.3 PROGRAM SEGMENT 1.3: NON-BUILT-IN FUNCTIONS IN OTHER SNOBOL4 IMPLEMENTATIONS

SPITBOL and related implementations provide a number of useful functions beyond the ones available in SNOBOL4, and these functions have been used freely in the programs. To run the programs under an implementation that does not have these functions therefore, the code for the functions must be provided. This segment provides the code for the additional functions used in these programs (and a few more). Normally, control skips around these function definitions so that they are not activated. To activate them, "comment out" or delete the go-to :(END1.2) found at the beginning of the segmen t. Certain additional functions, such as SETEXIT, which cannot be defined in the source language, have been avoided altogether.

6.2.1.4 PROGRAM SEGMENT 1.4: SPECS

The SPECS segment contains whatever SNOBOL4 statements the user may supply. It becomes an integral part of the program at run time. The user should therefore observe SNOBOL4 syntax rules. These statements are generally for overriding default values of program parameters. What parameters are relevant will depend on the particular control module being used. Also, Section 3.6 contains a master list of the parameters relevant to the built-in functions and to the various con trol modules.

6.2.2 PROGRAM SEGMENT 2: GENERAL INPUT FUNCTIONS

Actual reading from the input data file is done only in the READPG function. The input association, to the variable INFILE, was done in SEGMENT 1.1 of the program. READPG is the general-purpose hanging paragraph reading function. This function reads in a single hanging paragraph of text (concatenating the paragraph's records, if more than one, into a single string, PG.TEXT) and puts the id field, if any, of the first record of the paragraph into PG.ID. It does not care that the paragraph represents a band of a dictionary entry.

READ() on the other hand, is the band-reading function, and it does try to interpret the paragraph as a band. It calls READPG() to do the actual reading, then parses the resulting string PG.TEXT into the fields of a band and puts them into the four following variables, which are then available for other functions and the control module to look at, use, or alter:

HLEV — "headword level", the dots, or no dots (null string), before the band name or mode numbers.

MODE — the mode numbers, or lack of them (null string).

BAND — the band name, which is obligatory, being the first non-blank string in the paragraph after any dots and

numbers.

BODY — the body of the band, the rest of the paragraph after the band name.

In addition, READ() puts the PG.ID into the variable ID. The parsing is done by the global pattern PARAPAT, q.v., which may be overriden by the user in the SPECS file.

Actually, READ() does more than update these five registers. By calling on the function STEPREGISTERS(), it also keeps these five fields of the previous paragraph and those of the following paragraph in variables whose names are these names followed by .P (for previous) or .N (for next), e.g., ID.P, HLEV.P, MODE.P, ID.N, HLEV.N, etc. These registers are also available for inspection by other functons. Note that if a function, or the control module, alters the value of, say, BODY, then it is that altered value and not the original one that becomes the BODY.P when a new paragraph is read and parsed.

When READ() detects that there is no next paragraph, i.e. end-of-file has been reached, it sets the global flag EOFFLAG to 1. When called again, READ() will FRETURN. This flag can also be explicitly tested by a call to the predicate IFENDFILE().

READ() is affected by two global parameters which may be set by the user in the SPECS file, STOPAFTER and STOPPAT. STOPAFTER is normally null, and has no effect. However, if it is an integer, READ() will, after reading that many bands, behave as if it has hit end-of-file. Actually it will not necessarily stop right after that many bands, but proceed until (but not including) the next band that satisfies the pattern STOPPAT, q.v., which is defaulted to POS(0) '.' NOTANY('.'), i.e., the next main entry. This pattern may be overridden by the user in the SPECS file.

The function SKIPPG(N,PAT) may be used to skip up to the first paragraph that satisfies the pattern PAT after the Nth paragraph counting from the beginning of the file.

Among the other global variables that these functions use, these two may be of more general use: PG.RECCOUNT is a count of the records read by READPG(), and PG.COUNT is a count of the paragraphs read by READPG().

In summary, the functions in this program segment are:

```
READPG(N,PAT)
SKIPPG(N,PAT)
READ()
STEPREGISTERS()
IFENDFILE()
```

Of these functions only READ() and IFENDFILE() need be used by the control -module writer, and possibly SKIPPG(N,PAT). Global parameters of general interest are:

```
PARAPAT
STOPPAT
STOPAFTER
```

6.2.3 PROGRAM SEGMENT 3: GENERAL OUTPUT FUNCTIONS

The output functions are discussed in a discursive rather than a compartmental manner in order to better show how they fit together. They are:

```
SERFLD()
COUNTFLD()

LISTORWRITE(P,IDFLD)
LISTGEN(P,IDFLD)
WRITEBAND(P)
WRITEP(P,WIDTH)
PRINTP(P,DENT,OFFSET,WIDTH,SKIP,FOOT1,FOOT,HEAD,IDFLD)
BREAKP(P,L)

PRINTL(L,INDENT,SKIP,FOOT,HEAD,IDFLD)
CPAD(LINE,N)
```

These output functions have to do with output formating. They are not concerned with actual outputing itself, which is always done through calls to the WRITE() or PRINT() functions in SEGMENT 1 of the program. There are two kinds of formating. One is merely breaking up a band considered as a paragraph, according to hanging paragraph conventions and writing the resulting lines out. This may be done with an attached serial number field (e.g., for TSO EDIT files on the IBM 360) -- this format is called p-format (p for paragraph). The p-format is typically used only for storing dictionaries on disk or tape files for processing or on-line editing. This format can be read in again by the program, but is not particularly readable for the human reader.

The other kind of formating is the more complicated LISTGEN formating (l-format) used for hard copy printouts of dictionaries. The LISTGEN formating not only breaks up paragraphs into lines but also inserts indents, blank lines, page skips, title and page

number, entry number, paragraph number, etc. Each paragraph in this format still corresponds to a paragraph in the p-format. This format is meant to be more readable by the human reader, but there is no provision for the programs to read this format. The LISTGEN format is only a printing format; material in this format is not read in again.

BREAKP

The basic process common to both formating modes is the breaking up of paragraphs into lines of no longer than a given length in the so-called "hanging paragraph" format. The breaking up is done by calls to the function BREAKP(P,L), where P is the paragraph to be broken and L is the maximum length. When you need to break up a paragraph, call this function with that paragraph as the first argument and the length as the second. It will return the first line broken off P. Call it again with a null first argument to get the next piece. Continue this until it fails, signifying there is nothing left of the original P. Note that this function uses an "own" variable, BREAKPOWN, to keep the remains of the paragraph from one call to the next. BREAKPOWN is a global variable. (In SNOBOL4 there are only global and local variables, nothing in between.) Since it is global, you can work on only one paragraph at a time.

WRITEP

A simple example of using the BREAKP function is the WRITEP(P,WIDTH) function. It is short enough to reproduce here:

```
WRITEP   WRITE(BREAKP(P,WIDTH))          :F(RETURN)
WRITEP1  WRITE(' ' BREAKP(,WIDTH - 1))   :S(WRITEP1)F(RETURN)
```

The first call to BREAKP presents it with the paragraph, P, to be broken, and the resulting first piece, if any, is passed to the WRITE function to be written out, possibly with a serial number attached. (See WRITE(L) in program SEGMENT 1, in Section 6.2.1.1 of this chapter). In subsequent calls (in the loop formed by the second statement) BREAKP is presented with a null first argument, meaning additional pieces from the same paragraph will be returned. But the maximum length of these pieces will be one shorter than that of the first piece, due to the decremented value of the second argument. This is to allow the initial blank (which signals continuation lines in the "hanging paragraph" conventions) to be attached without overstepping the maximum allowed width (the WIDTH argument of WRITEP) of the lines.

WRITEBAND

WRITEP is called by WRITEBAND(P), the function which writes out a band in p-format. The line width which it passes to WRITEP(P,WIDTH) is the value of the global variable LINESIZE, which has a default value of 70. This may, of course, be overridden by the user in the SPECS file. Seventy is a reasonable maximum width for lines in disk

files that are to be edited at a terminal, most screens being able to display 80 characters across, some of which may be taken up by a line number. The band which WRITEBAND writes is its argument, if the latter is non-null. In this case, the argument consists of the entire band (including band label) as a single string. However, if the argument is null, the function assembles the band from the current values of the four variables HLEV, MODE, BAND, and BODY, and writes it out.

LISTGEN

Another function which calls BREAKP(P,WIDTH) is LISTGEN(P,IDFLD), which is the function that prints out a band with indentations and other formating, in the so-called l-format. (Note that there is also a control module called LISTGEN.) In contrast to the other output functions discussed so far, LISTGEN does print formatting. It is, however, analogous to WRITEBAND in that it takes a band, which is either its P argument or, if that is null, the current band assembled from the values of HLEV, MODE, BAND, and BODY, and outputs it. LISTGEN however, is much more complicated, since it has to recognize entry levels (main vs. sub-entries, etc.), mode levels, boundaries between entries and modes, etc. in order to insert indentations and line skips in the output. Because they affect the maximum line length of the paragraphs, the indentations are passed down ultimately to BREAKP. By consulting the flag, SUPBANDLABEL, LISTGEN also determines whether the band label (that is, the HLEV, MODE, and BAND fields) is to be printed. It also updates the page heading (but does not print it--that's PRINTL's job) whenever a headword, a one-dot, band comes along. LISTGEN's second argument, IDFLD, is taken as an id field attached to the band. LISTGEN does not care what it consists of, but the calling program should make it of consistent length from one call to the next of LISTGEN. The general idea of LISTGEN is that it does only those decisions that are oriented to band-formating. LISTGEN calls PRINTP to do further print-oriented things. See the control module LISTGEN for an example of how the function LISTGEN can be used.

PRINTP

PRINTP takes a paragraph, its first argument, and breaks it up using BREAKP into lines, then prints the lines by calling PRINTL. It manages, generally, the horizontal positioning of the paragraph. The full list of arguments is shown here:
PRINTP(P,DENT,OFFSET,WIDTH,SKIP,FOOT1,FOOT,HEAD,IDFLD). IDFLD is the id field to be printed at the left of the first line of the paragraph, but not of subsequent lines. What the other arguments mean is shown in the layout chart in the program listing.

PRINTL

PRINTL(L,INDENT,SKIP,FOOT,HEAD,IDFLD) prints the IDFLD and line L, with INDENT spaces separating them, after skipping SKIP blank lines. The IDFLD is right up to the left margin, determined by the

global variable LMARGIN. The main job of this function is to keep track of the current vertical position on the page, through the variable LINESLEFT, and to skip a page at the appropriate place. The function keeps track of the page count and prints it as well as the current running head (the argument HEAD), right justified, and title (the global variable TITLE), centered. The lines per page is the global variable LINESPERPAGE, defaulted to 60, the heading can be suppressed by setting SUPHEADING to a non-null value. The actual writing out is, of course, done by calling the output functions PRINT and PRINTPAGE of SEGMENT 1.1.

CPAD

The function CPAD(LINE,N) returns its first argument centered, padded with blanks on both sides to a total length of N. It is used by PRINTL for printing page titles.

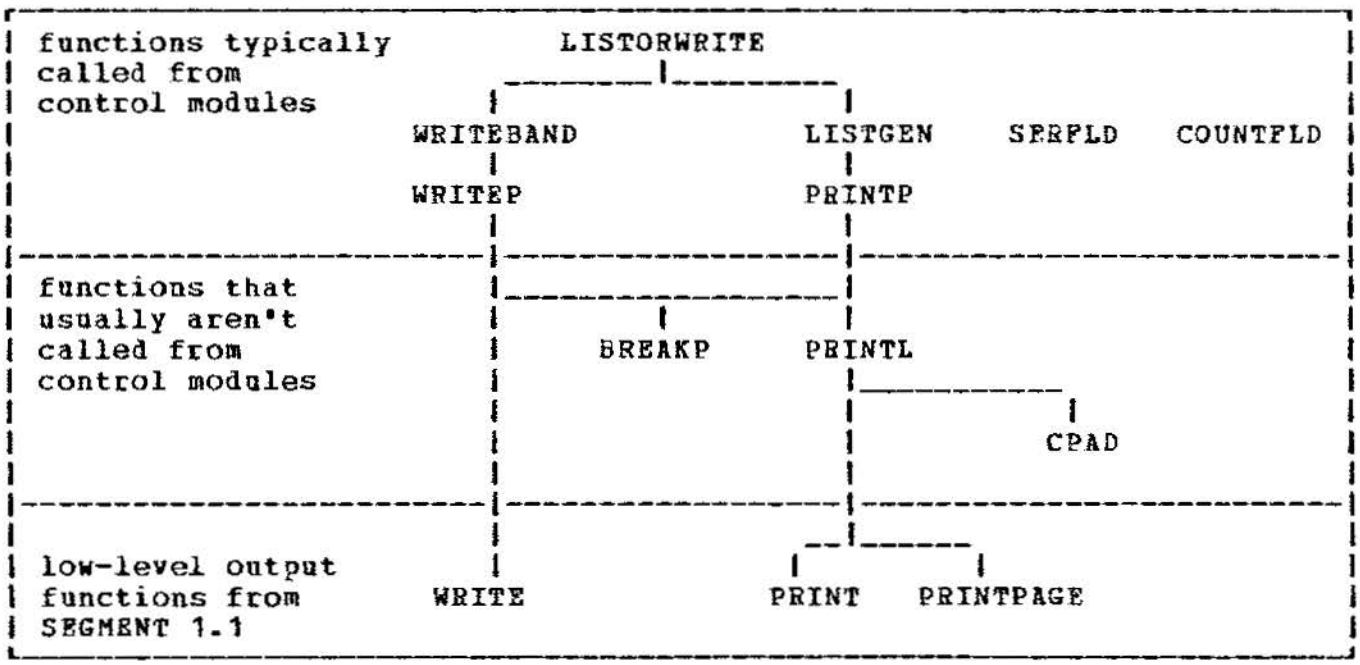
LISTORWRITE

In control modules that produce band-format output, it is sometimes convenient to not build in the decision to output the bands in p-format (for storage) or l-format (for listing), but to make this depend on a flag that can be easily set by the user in the SPECS file. Such control modules can call LISTORWRITE(P,IDFLD) instead of WRITEBAND or LISTGEN directly. LISTORWRITE will then call one or the other of these depending on the flag PFORMAT. This switch is defaulted to null, which means that LISTGEN, giving l-format output, is the default. If the user wants p-format, set PFORMAT to a non-null value in the SPECS file. (This option is not currently available in all the control modules.)

SERFLD, COUNTFLD

The functions SERFLD() and COUNTFLD() are not called by any functions within this SEGMENT, but are used by control modules such as LISTGEN (not the function LISTGEN but the control module of the same name), to generate various numbers to put in the id field in LISTGENed output. SERFLD() gets the current serial number field from the input file if there is any and if the flag SNUM is null. COUNTFLD() keeps count of the headword bands by looking in HLEV and returns the current count if the current band is a main headword band. (The way these operate and interface with the rest of the functions is not very flexible or logical and needs to be re-designed.)

SUMMARY OF RELATIONS AMONG THE OUTPUT FUNCTIONS



6.2.4 PROGRAM SEGMENT 4: BAND PROCESSING FUNCTIONS

These functions do the actual testing and processing of dictionary materials. Generally they act on one band at a time, that is, they assume there is a "current" band, the band which has been read in by READ() and parsed into the registers HLEV, BAND, MODE, and BODY. They also have access to the "previous" and "next" bands in the registers HLEV.P etc. and HLEV.N etc. respectively (see Section 6.2.2).

The following is a classified list of the functions.

BAND Predicates

```
IFBANDS(BANDS)
IFLITBANDS(BANDS)
IFBANDQ()
IFENTRY()
IFENDENT()
IFMAIN()
IFENDNEST()
IFENDFILE()
IFSUB()
IFNEWMODE()
IFMATCH(PAT)
```

KEEP-related functions

```
CLEARKEEP()
KEEPBAND()
REREAD()
IFENDKEEP()
RECONTROL()
REVIEW()
SAVEPOINTER(INCH)
RESETPOINTER()
REREADREGS()
WRITEKEEP()
```

STATE-setting and testing functions

```
SET(STATENAME,STATEVAL)
UNSET(STATENAME)
IF(STATENAME,STATEVAL)
STATE(STATENAME)
```

PACKET-related functions

```
CLEARPACKET()
PACK(STUFF)
LISTPACKET(ENT)
```

Miscellaneous functions

```
BANDAID(BANDLIST)
SKIP()
```

6.2.4.1 PROGRAM SEGMENT 4.1: BAND PREDICATES

The first group of functions, in SEGMENT 4.1, are predicates which test for various "positions" in an entry. They succeed if the current band is at such a position and fail otherwise. Most of them are extremely simple tests that can be replaced by single calls to built-in predicates that look at the various registers like HLEV etc. However, in writing a control module, it might be more mnemonic to use these functions instead. Predicates, and only predicates, have names beginning with 'IF'. Another convention that has so far been followed is that predicates cause no side effects and return no value. Other functions may cause side effects and return values but do not fail, so that predicates vs. non-predicates form two disjoint classes of functions with respect to these properties. (Currently the READ() function violates this convention in that it can fail. It will be corrected.)

IFENTRY() checks if the current band is the beginning of an entry (either main or sub), IFMAIN() checks if the current band begins a main entry, IFSUB() checks if the current band begins a subentry of any level, IFENDENT() checks if the current band is the last band of an entry (of whatever level), and IFENDNEST() checks if the current band is the last band in a "nest" of entries, i.e. that the next band, if any, is a main entry band. IFENDFILE() checks if the end

of the input file has been reached. IFNEWMODE() checks if the mode of the current band is different from that of the previous band.

IFBANDS(BANDS) checks if the current band is one of the bands given in the argument list BANDS in the form, 'a,b,c,d'. It is indifferent as to upper or lower case, that is, the band names in the dictionary can be in upper or lower case and they also can in this argument list. However, IFLITBANDS(BANDS) is case-sensitive, that is, it will consider 'a' and 'A' as two different band names.

IFBANDQ() also is a test for bands, but it takes no arguments. Instead, it uses the global variables BANDS and NOTBANDS (which are strings of bandnames), and ALLBANDS and NOBANDS (which are flags). It succeeds or fails as follows: If NOBANDS is set (is non-null) when it is first called, it will fail and always thereafter fail, no matter what the current band is; otherwise if ALLBANDS is set when it is first called, it will succeed and always thereafter succeed; otherwise if BANDS is non-null when it is first called, it will always use the bandnames currently in BANDS, succeeding if the current band is among them and failing otherwise; otherwise if NOTBANDS is non-null when it is first called, it will always use the bandnames currently in NOTBANDS, succeeding if the current band is in that list and failing otherwise; otherwise, i.e., if all four variables are null when IFBANDQ() is first called, then it will succeed and always thereafter succeed. IFBANDQ(), like IFBANDS(BANDS), is not sensitive to upper/lower case in bandnames.

IFMATCH(PAT) simply does a pattern match on the current BODY using the argument PAT, and succeeds and fails according to the success or failure of the match.

6.2.4.2 PROGRAM SEGMENT 4.2: KEEP-RELATED FUNCTIONS

Since READ() reads in only one band at a time, the amount of a dictionary that can be examined at any time is limited. Specifically, it is limited to the current band, the band before the current band and the band after. A surprising amount of useful things can be done under this myopic restriction, especially by setting and testing flags (SEGMENT 4.3). In fact, most control modules do not look beyond this range. This is because many decisions about what to do with a band can be made without looking beyond this neighborhood or only by looking back, not forward, in the entry. Looking back can be accomplished by setting flags when certain conditions come along, in case this information will be needed later in the entry. When, later in the entry, these flags are tested, this constitutes "looking back", even though the bands or conditions that caused those flags to be set are no longer around to be looked at. When "looking forward" in an entry is required to decide what to do with a given band, a control module has to READ() ahead and hence must have a place to store the intervening bands.

A storage facility for such occasions is provided in the array KEEP and associated functions. The function CLEARKEEP() clears it. The function KEEPBAND() adds the current band to it. The function WRITEKEEP() writes out the entire KEEP using the function WRITEBAND. The function REVIEW(), which can be called in the control module, re-reads the KEEP, band by band, calling the function RECONTROL(), which has the entry label REVIEW, for each band. Therefore, there should be in the control module a statement labeled REVIEW containing instructions for what to do for each re-read band. The instructions should end by transferring to RETURN. If it transfers to FRETURN, the re-reading of KEEP is aborted. The functions REREAD() and IFENDKEEP() support the REVIEW function and need not be called explicitly from the control module. The pair of functions SAVEPOINTER() and RESETPOINTER() saves the current KEEP position used by KEEPBAND, and resets the current KEEP position to the saved value, respectively. The control module ONBANDS is an example of using the KEEP facility. (As this is a rather awkward facility to use, a better way is being sought.)

6.2.4.3 PROGRAM SEGMENT 4.3: STATE-SETTING AND TESTING FUNCTIONS

A "state" is simply an entry in a special table, STATETABLE, and can be assigned any value. It may be used like an ordinary variable, except that there are special functions to manipulate states conveniently. To set state "A" to a value "X", call SET('A','X'). If the second argument is null, a value of 1 is used. In this way a state can be used as a flag. To unset a state, i.e. to reset it to null, call UNSET('A'). To test if it is non-null, call the predicate IF('A'). To test if it has the value 'X', call IF('A','X'). The value to which it is set can be gotten as the value of the function STATE('A'), which is equivalent to STATETABLE<'A'>.

6.2.4.4 PROGRAM SEGMENT 4.4: PACKET-RELATED FUNCTIONS

PACKET is a special variable which can be manipulated with certain functions. CLEARPACKET() clears it. PACK(X) concatenates its argument to it. LISTPACKET() calls LISTGEN on the PACKET.

6.2.4.5 PROGRAM SEGMENT 4.5: MISCELLANEOUS FUNCTIONS

When BANDAID(BANDLIST) is called, it checks to see if the current band name (in the variable BAND) is in the argument list. If it is not in the list, the function simply returns. If present, it transfers control to a user statement with a label consisting of the band name, in upper case, followed by a single period. The user must, therefore, provide such statements, most likely in the SPECS file. The statement must eventually transfer either to RETURN or to SKIP. An alternative to transferring to SKIP is to call the function SKIP() and then transferring to RETURN.

6.2.5 PROGRAM SEGMENT 5: SORT I/O FUNCTIONS

Segment 5 of the program does not occur in a simple BIRDBATH run, one which does not involve sorting. It is added to the normal BIRDBATH segments to construct a TOAST phase 1 or phase 2 program. It contains functions which write sort records in a standard format and functions which read the sorted records back in and check for "control field breaks". The functions are:

phase-detection function

```
DETPHASE()
```

Functions for Phase 1

```
WRITESORT(REC,H1,H2,H3,H4,H5,H6,H7,H8,H9,H10,H11,H12,H13,H14)
WRITESORTCNTL()
HANDLE(HANDLE)
HANDLEB(HANDLEB)
```

Functions for Phase 2

```
READ()N
STEPFIELDS()N
IFBREAK(N)
IFENDBRK(N)
```

The function which writes out sort records is WRITESORT(REC,H1,H2,..), where REC is a record to be sorted, and H1, H2, etc. are "handles", or sort fields to be used, in the order given, for sorting the record. These handles are considered separate from the data record, because in almost all cases of sorting, the desired sort sequence is not directly related to any part of the data record but can only be stated in terms of fields generated from the data record (or from other material) by means of a non-trivial function. For convenience two simple such functions are provided in this segment: HANDLE(X) and HANDLEB(X). The actual number, NHANDLES, of distinct handles, H1, H2, etc. that will be given as arguments whenever WRITESORT is called in the control module should be specified in a statement in the control module, e.g. NHANDLES = 4. This number is defaulted to 14, but should be overridden for efficiency reasons if in fact a smaller number is used (as will normally be the case).

The significance of giving WRITESORT a number of separate handles, rather than stringing them together into one handle (which is also possible), is that WRITESORT will keep them separate by inserting a special character, SEP2, between them. Then, when the sorted file is read in, these fields can be recognized by the READ() function. Changes in their contents ("control field breaks") can be used to trigger output formating or other processes according to what was the highest level (H1, H2, etc.) in the hierarchy of fields the change occurred in. The detection of the beginning or end of a group of records with the same handle of a given level is very much like testing for the beginning or end of an entry when the input file is a band-format file rather than a sort output file. Here the

detection is done with the functions IFBREAK(N) and IFENDBRK(N), where N is the level (1 for highest) of the handle being tested. IFBREAK(N) succeeds if and only if the current record has any level-N and higher handles that are different from the corresponding ones of the previous record. IFENDBRK(N) succeeds if and only if the current record has any handles of level-N and higher that are different from the corresponding ones of the next record. The logical sequence in which to test for these conditions without missing any is IFBREAK with increasing N followed by IFENDBRK with decreasing N, as in the following schema of a phase 2 control loop:

```
PHASE2    IFENDFILE()  STOP()
          READ()
          IFBREAK(1)   ...
           IFBREAK(2)   ...
            IFBREAK(3)   ...
               .........
            IFENDBRK(3)  ...
           IFENDBRK(2)  ...
          IFENDBRK(1)  ...          :(PHASE2)
```

Note that the function READ() is used also in phase 2, but it is a slightly different function from the phase 1 function of the same name. The details need not concern the writer of control modules. The main point to remember is that it leaves the data record in the variable REC. This should be the same string that was given as the first argument to WRITESORT, in phase 1, to write out for sorting. (STEPFIELDS() is a function used by the Phase 2 READ(); it need not concern the user.)

Both phase 1 and phase 2 functions are in this single segment, SEGMENT 5. This is solely for ease of maintenance and documentation. The program listing makes a little more sense when seen as a whole or as two halves of a whole rather than separately. Which set of functions is to be activated (DEFINEd) is determined after a call, during initialization of SEGMENT 5, to the function DETPHASE(). This function takes a peek at the first record of the input file (and then rewinds it) to see if this should be a phase 1 or 2, and returns this number. During initialization, this number is assigned to the variable PHASE, which can also later be checked by the control module to decide which phase's control loop it should enter.

6.3 WRITING NEW CONTROL MODULES

The control module, SEGMENT 7, in any BIRDBATH run is the "executive", or "driver", that directs things to happen. All the previous segments merely set up functions, parameters, etc., that may be used by the control module. These functions perform rather low-level, general operations and tests, which are closely tied to the physical representation of the dictionary entries. More specialized functions can be included at the front of the control module needing them or can be included in SEGMENT 6, the "User Functions", if they are of somewhat more general applicability. Even though the set of available low-level functions is the same from run to run, the runs can be quite different because of the different control modules.

A typical control module is a loop which calls the READ() function to read in a band and parse it into its different fields, HLEV, MODE, BAND, and BODY. Then it might do some tests such as IFMAIN() or IFBANDS(some band names) or IFENDENT(), and perform some actions such as PACK(something), or set some flags, or execute some pattern match and replacement on the BODY. Then there might be a call to some output function such as LISTGEN or WRITEBAND. The loop is controlled by the test IFENDFILE() followed by STOP(), most conveniently placed right at the beginning of the loop. (There need not be an explicit transfer to END because there is such a transfer within the function STOP().) However, the control module, being the very last segment of the program, must end with the standard SNOBOL4 'END' statement. A very simple such loop might look like the following:

```
LOOP    IFENDFILE()   STOP()
        READ()
        IFBANDS('hw,df')  LISTGEN()   :(LOOP)
END
```

Since the control module operates in the same environment as all the lower level functions, and since SNOBOL4 also has few controls and devices for isolating program contexts, there is the danger, in this kind of program organization, of unknowingly using the same variables or labels already built in. Such collisions are not as serious with labels as with variables since duplicate labels are detected and flagged by the compiler and an error message is printed in the compilation listing.

Unknowing use of built-in BIRDBATH variables is more serious, however, since the system has no way of telling that you are not really referring to the same variable. The way to avoid duplicate variables is to not use ordinary variables when writing control modules but to use the "state-setting functions" (SEGMENT 4.3), which essentially give access to a separate variable space through the functions SET, UNSET, STATE, and IF. This variable space is merely a TABLE, called STATETABLE. It may be accessed with the usual table reference, e.g., STATETABLE<X>, but calling the

functions is often simpler and does not require a separate statement, and can result in a saving of statements and of go-to's. This variable space is reserved strictly for the user, and is not used by the system of built-in functions. (The current exception is the table argument 'SKIP', which is used by the built-in function BANDAID(). This will be corrected in the future.)

There are some built-in variables that the user (the writer of control modules) may legitimately need to access, for example, in order to do a pattern match on the BODY of a band. A preliminary list of these is given later in this section. It will be filled out with a description of what they are. A preliminary list of built-in labels is also given. A complete index of global variables and functions is now available in Section 6.4. It should be used in conjunction with a program listing.

A TOAST control module contains two parts, a phase-1 and a phase-2. At the beginning of the module there must be a test of the global variable PHASE, to see if the module is being executed in phase-1 or phase-2 of the TOAST run, with a transfer to the appropriate portion of the control module. In addition, there should be, before this transfer, an assignment to the variable NHANDLES. This should be the number of sort handles the WRITESORT(REC,H1,H2,...) function will be called with in phase-1. A statement should also be placed here, for use by phase-1, that allows the user to supply a handle function having any name, not necessarily 'HANDLE'. Note that 'HANDLE' is in quotes; HANDLENAME is not.

Otherwise, each of the two portions of the control module can look quite like a BIRDBATH control module. Figure 6.1 shows a typical schema of a TOAST control module.

```
        NHANDLES = 2
        DIFFER(HANDLENAME) OPSYN('HANDLE',HANDLENAME)
        EQ(PHASE,1)                :F(LOOP2)
LOOP1   IFENDFILE() STOP()
        READ()
        .
        .
        .
        WRITESORT(REC,H1,H2)       :(LOOP1)

LOOP2    IFENDFILE()   STOP()
         READ()
          .
          .
          .
         WRITEBAND()                :(LOOP2)
END
```

Figure 6.1

Note that the READ() function is called also in phase-2. This is however a different function from that defined for phase-1. It does not read bands but rather reads the sort records originally written out by WRITESORT and since sorted. Of course, it also does not parse the records into the band fields such as HLEV and MODE but rather into handle fields. Changes in these handle fields from one record to the next can be detected by calls to the predicates IFBREAK(N) and IFENDBRK(N) The first one checks for a control break of order N. A control break of order N occurs whenever the Nth handle of the current record, or any handle to the left (N-1, N-2, etc.), is different from the corresponding handle of the previous record. (This implies that whenever a control break of order N occurs, a control break of order N+1 also occurs.) IFENDBRK(N) checks if the next record would cause a control break of order N, or if there is no next record (end of file). Phase 2 loops are driven by these control breaks rather than by the logical breaks in a dictionary file. More comments may be found in PROGRAM SEGMENT 5, SORT I/O FUNCTIONS, and in Section 6.2.5 of the manual. Listings of the existing TOAST control modules should also be studied.

The comments about writing BIRDBATH control modules also apply here. The two loops of the TOAST control module, however, operate in entirely different environments since they are separated in time by the sort step and since the phase-1 program is terminated before the sort is performed. The only way the two phases communicate with each other is through the sort file.

The following list contains only BIRDBATH labels and names, but no SNOBOL4 or SPITBOL built-in labels and names. Also, this list gives only those labels and names that are 6 characters or shorter. Section 6.4 gives a complete list of built-in variables and functions.

Labels

```
CPAD
IF
IFSUB
PACK
PRINT
PRINT1 through PRINT4
PRINTL
READPG
REREAD
SET
SKIP
SKIPPG
STATE
UNSET
WRITE
```

Variables

```
BAND, BAND.N, BAND.P
BINDEX
BODY, BODY.N, BODY.P
EOF
HANG1
HLEV, HLEV.N, HLEV.P
ID,   ID.N,   ID.P
IDLEN
INFILE
LC
LP2
MAXLEN
MODE, MODE.N, MODE.P
OUTFILE
PG.ID, PG.ID1, PG.ID2
PRINTER
TITLE
UC
```

6.4 INDEX OF VARIABLES AND FUNCTIONS

Below is an index of global variables, functions, and operators in the built-in segments (SEGMENTS 1 through 5) and in the existing control modules. This list may not be up-to-date.

6.4.1 VARIABLES AND FUNCTIONS IN BUILT-IN PROGRAM SEGMENTS

6.4.2 VARIABLES AND FUNCTIONS IN CONTROL MODULES

6.5 INSTALLATION NOTES

For transportation to other installations, the programs are put on an IBM (EBCDIC-coded) 9-track unlabeled tape, 1600bpi unless otherwise noted. There are some 22 program files, depending on how many control modules have been included. The files are mostly quite short, on the order of 100 records; a few have just a handful of records; one or two have several hundred. They are all fixed-length 80-character card-image records with no sequence numbers. Blocksize is 4000 unless otherwise noted. They consist of source-language (SNOBOL4 and Spitbol) programs, as described below.

In addition, there may also be a long text file on the tape consisting of this manual, formatted by the SCRIPT text formatter, complete with carriage-control codes and ready for listing on a line printer. Format is variable-length records, blocked: RECFM=VB,LRECL=137,BLKSIZE=4000. There are some 230 blocks.

Please read Section 6.1 of this manual before trying to understand the following. Files 2, 3, and 4 correspond to program "layers" 1, 3, and 4 as described in Section 6.1. At the University of Hawaii (UH), they are stored in files called BIRDBATH.INIT.DATA, BIRDBATH.BUILTIN.NEW.DATA, and TOAST.SORTIO.DATA, respectively, as shown in the catalogued procedures (files 21, 22).

File 1 is a very short file consisting mostly of -COPY control cards, which in Spitbol cause the other source files to be copied in. It, along with the Job Control Language statements (JCL), is the means, on the IBM 370 installation, by which the different "layers" of the program sandwich are assembled for each run. At UH it is stored in a file called BIRDBATH.ROOT.DATA. This file does not contain any program, and may be dispensed with if some other means of assembling the program layers is used.

Files 5 through 13, unless otherwise noted, contain control modules for BIRDBATH, as listed below. At UH they are stored in members of a partitioned dataset, as BIRDBATH.CONTROL.DATA(BANDAID), BIRDBATH.CONTROL.DATA(BANDFILT), etc.

Files 14 through 20, unless otherwise noted, contain control modules for TOAST, as listed below. At UH they are stored in members of another partitioned dataset, as TOAST.CONTROL.DATA(BANDSORT), TOAST.CONTROL.DATA(HANDSORT), etc.

Files 21 and 22, unless otherwise noted, are copies of the two catalogued procedures, BIRDBATH and TOAST, as installed on the IBM/3081 (running MVS) at UH. They may provide some guidance in installing these programs on a similar computing system.

Installing the programs means not only copying the program files from tape onto a particular computing system but also setting up procedures so that BIRDBATH and TOAST runs could be launched by the user with a minimum of irrelevant ritual. For the simplest runs the

user should have to specify, apart from accounting information, little more than the control module, the SPECS file or the SPECS statements themselves, the input file, and the output file if any. The basic requirements for assembling the program components for a BIRDBATH or a TOAST run are set out in Section 6.1. A different operating system would require different methods of assembling a run than those shown in Files 21 and 22.

File 23, unless otherwise noted, is the formatted manual, with control codes, ready for printing on a line printer with upper-and-lower case print train. It is formatted for 8 1/2" x 11" paper. On an IBM system it may be printed with the IBM utility program IEBPTPCH, using JCL similar to the following:

```
//     EXEC  PGM=IEBPTPCH
//SYSPRINT  DD  SYSOUT=A
//SYSUT1    DD  UNIT=TAPE,VOL=SER=LIN115,DISP=OLD,LABEL=(23,NL),
//  DSN=LIN115,DCB=(RECFM=VB,LRECL=137,BLKSIZE=4000,DEN=3)
//SYSUT2    DD  SYSOUT=(A,,260)
//SYSIN     DD  *
     PRINT  PREFORM=A
//
```

(The "260" on the SYSUT2 DD statement is a local forms code, for TN train and 8 1/2 x 11 unlined paper.)

The following are the files on tape LIN115 at the time this is written. If a different tape with a slightly different inventory of files is supplied, a similar list will accompany that tape, and any deviations from the above descriptions will be noted.

file number	description (all except the last file have fixed length (80) records, blocked 4000)
1	"root" of program, with -COPY statements to assemble program
2	INIT layer of program (initializations)
3	CORE layer of program (built-in functions etc.)
4	SORTIO layer (for TOAST runs only)
	Control modules for BIRDBATH runs:
5	BANDAID
6	BANDFILT
7	BANDPACK
8	EDLISTB
9	LISTGEN
10	NULLRUN
11	ONBANDS
12	ONBANDSR
13	STATS

Control modules for TOAST runs:

```
14      BANDSORT
15      HANDSORT
16      INVERT
17      INVERT2
18      NULLRUN
19      REBANDLE
20      XCHECK
```

Copies of catalogued procedures:

```
21      BIRDBATH
22      TOAST
```

LEXWARE Manual (variable blocked: lrecl=137,blksize=4000):

```
23      Formatted text file.
```

6.5.1 INSTALLING ON AN IBM MAINFRAME RUNNING OS

These notes assume that the 360 or 370 Spitbol compiler is being used.

Copy the first 4 files on the tape into the following datasets, with an appropriate account prefix:

```
BIRDBATH.ROOT.DATA,
BIRDBATH.INIT.DATA,
BIRDBATH.BUILTIN.NEW.DATA,
TOAST.SORTIO.DATA
```

Copy the BIRDBATH control modules (files 5-13 in above list, or corresponding files if the tape you received is different) into members of a partitioned dataset, BIRDBATH.CONTROL.DATA.

Copy the TOAST control modules (files 14-20, mutatis mutandis), into members of a partitioned dataset, TOAST.CONTROL.DATA.

Copy the two catalogued procedures from the tape into datasets for editing. In these procedures change the account prefix 'T119920' on all the dataset references to the account prefix where you are storing the above program segments. Check disk volume references and disk BLKSIZE parameters and adjust for compatibility with your installation. Adjust method of referencing forms code and COPIES code in //MPAPER DD and //MDUMP DD statements. Check through for remaining incompatibilitie, and adjust for your installation.

Test each procedure as an in-stream procedure (you have to add a // PEND statement) first with "nullruns":

```
//    EXEC    BIRDBATH,CONTROL=NULLRUN
```

and

```
//    EXEC    TOAST,CONTROL=NULLRUN
```

to see if source programs are in place (should result in compilation listings and "normal termination" messages). Then test using test data and one of the recipes from Section 4.2 and one from Section 4.3.

6.5.2 INSTALLING ON A DEC-2060 RUNNING TOPS-20

These notes assume that the SITBOL compiler, called by the command ITBOL, is available.

Ignoring the first file on the tape, copy files 2, 3 and 4 into corresponding DEC-20 disk files under your account, giving them names as follows:

2	INIT.SNO
3	CORE.SNO
4	SORTIO.SNO

Copy the BIRDBATH and TOAST control modules from the subsequent tape files into corresponding disk files. See the list of the tape files given near the begining of 6.5. Since SITBOL expects file names to be no longer than 6 characters, not counting the extension (this is a DEC-10 and TOPS-10, not a TOPS-20, convention, but SITBOL was written for the 10), some abbreviating of the control module names will be necessary:

BIRDBATH CONTROL MODULES

tape file number	DEC-20 disk file name
5	BNDAID.SNO
6	BDFILT.SNO
7	BDPACK.SNO
8	EDLSTB.SNO
9	LSTGEN.SNO
10	NULRUN.SNO
11	ONBNDS.SNO
12	ONBNDR.SNO
13	STATS.SNO

TOAST CONTROL MODULES

14	BDSORT.SNO
15	RDSORT.SNO
16	INVERT.SNO
17	INVRT2.SNO
18	NULRUN.SNO
19	REBNDL.SNO
20	XCHECK.SNO

Check for any character-set incompatibilities in the source files. Toward the end of the INIT file, find program SEGMENT 1.3. It contains some comments about special characters that may need to be changed to conform with the host computer. Refer to the SITBOL manual, section 1.4.2, Program.

Modify the INPUT and OUTPUT associations. Near the beginning of the INIT file, in program SEGMENT 1: INITIALIZATIONS, locate two statements beginning with INPUT('INFILE','INFILE'). Replace them with these two statements:

```
INPUT('INFILE','INFILE:')          OUTPUT('OUTFILE')
OUTPUT('MESSAGE','MSG.LST',' ')    OUTPUT('MESSPAGE','MSG.LST','1')
```

Adjust certain statements in the SORTIO.SNO program module:

1. In the DETPHASE function (this is the first label in the module, change the REWIND function call to REWIND('INFILE:'), i.e., add the colon.

2. In the PHASE1INIT function (a couple of labels further down) increase MAXSORTLEN from 255 to some large number like 100000.

3. After the label WRITESORT5 (about 50 lines further down) replace the long call to the WRITE function (that takes up two lines) with these three statements:

```
    SORTREC = HANDLE LPAD(RECSEQ,6,'0') SEP2 LPAD(SUBQ,2,'0')
+             SEP1 REC1
    SORTRECSIZE = GT(SIZE(SORTREC),SORTRECSIZE) SIZE(SORTREC)
    WRITE(SORTREC)
```

(Those are zero's in the quotes.)

4. In the WRITESORTCNTL function (some 15 lines further down) change the OUTPUT association statement and the next statement to:

```
   OUTPUT('SORTCNTL','SRTCTL.TMP')
   SORTCNTL = 'SORT/RECORD-SIZE:'  SORTRECSIZE  '/KEY:1,'
+            MINSORTLEN  '  SORTIN.TMP  SORTOU.TMP'
```

To run a BIRDBATH run, first put into a file called SPECS.SNO any desired SPECS statements, then construct and submit the following batch job. In this job you need to supply the name(s) of your input file(s), of the output file, and of the control module.

```
efine  infile:  <input data file stream>
itbol
*<output file name>.dat,tty:,tty:=init,specs,core,<control module>
f(noerror)  oto end
*<circumflex>c
%err::
end::
rint msg.lst/file:fortran
%fin::
```

If <output file name>.dat is to be listed (not all control modules generate listable output files), examine it first if desired, then print it.

Explanation:

In the first statement, <input data file stream> should be one or more data file (dictionary file) names, separated by commas if more than one.

The line after the itbol command gives the SITBOL compiler the names of output and input files in the format it expects. To the left of the equals sign are three standard output files: the first is for the program-generated output, the second is for the compilation listing, the third is for SITBOL error messages. We here send the last two to TTY:, which for a batch job is automatically listed (printed on hardcopy). After the eqials sign is the input file stream to the compiler. The extension .SNO is assumed by the compiler if no extension is given (hence all program modules were named with that extension).

The f(noerror) command operates together with the %err:: and end:: labels to insure that the last command is executed whether or not SITBOL terminates with an error.

The <circumflex> (or "up-arrow") symbol followed by "c" exits from the compiler.

The last command, rint msg.lst/file:fortran, sends the message file generated by the BIRDBATH program (the "BIRDBATH RUN LOG") to the printer. The lst:fortran causes the listing to respect carriage-control codes. Being a .lst file, it is deleted after printing.

To run a TOAST run, first put into a file called SPECS.SNO any desired SPECS statements, then construct and submit the following batch job, in which, again, you need to specify the name(s) of your input file(s), the name of the output file, and the name of the control module. The blank lines are inserted only for clarity; they are not part of the job.

```
efine  infile:  <input data file stream>
itbol
*sortin.tmp,tty:,tty:=init,specs,core,sortio,<cmodule>
*<circumflex>c

ename  msg.lst  msg1.lst
ort
*take  srtctl.tmp
*<circumflex>c
*delete  sortin.tmp
xpunge

efine  infile:  sortou.tmp
itbol
*<output file name>.dat,tty:,tty:=init,specs,core,sortio,<cmodule>
f(noerror) oto end
*<circumflex>c
%err::
end::

rint msg1.lst,msg.lst/file:fortran
elete srtctl.tmp,srtctl.log,sortou.tmp
%fin::
```

After the job has run, examine <output file name>.dat and print it if appropriate.

Some explanation:

A TOAST run consists of two BIRDBATH runs with a sort sandwiched in between, with some passing of files from one step to the next. The first BIRDBATH program passes the file SORTIN.TMP to the sort. It also makes the short file SRTCTL.TMP, consisting of the sort control statement, to be given to the sort by the *take command. After the sort, the sortin.tmp file is deleted and expunged to free up some disk space. The SRTCTL.TMP file instructs the sort to use SORTIN.TMP as the input file and SORTOU.TMP as the output file. The latter file is then defined as the input file to the second BIRDBATH program, which otherwise looks like a regular BIRDBATH run. Since both BIRDBATH programs write their Logs to MSG.LST, this file is renamed after the first program so as not to be overwritten by the second, then both are printed at the end. Finally, some temporary files are deleted. One of these, SRTCLT.LOG, is made by the sort program, but a copy of it is also sent by the sort program to TTY:, so it does not have to be explicitly printed.

These are of course not the only possible ways to set up BIRDBATH and TOAST runs. There are other ways to associate files, concatenate them, and pass them from one program to another. However the ubmit command does not have the capability of passing parameters into the batch file being submitted. Nor is there a facility corresponding to MIC on the DEC-10, which allows macro commands with arguments. Therefore each time the batch file is to

be submitted, while it may not have to be typed in from scratch, it may still be necessary to first edit into it the names of input files, control module, and output file.

6.5.3 INSTALLING ON A PDP11 OR VAX RUNNING UNIX

These notes assume that the Macro Spitbol compiler, called by the command spitbol, is available.

Copy tape files 2, 3, 4 into corresponding files in a subdirectory lex/ as follows:

```
2           lex/init
3           lex/core
4           lex/sortio
```

Copy the BIRDBATH control modules into corresponding files in a subdirectory lexb/:

```
5           lexb/bandaid
6           lexb/bandfilt
7           lexb/bandpack
    etc.
```

Copy the TOAST control modules into files in a subdirectory lext/:

```
14          lext/bandsort
15          lext/handsort
    etc.
```

1. In the first file, lex/init, find SEGMENT 1.3. There are some comments about characters that may have been converted during the copying. If, for instance, the vertical bar described in the comment shows up as an exclamation mark next to it, it will have to be converted back to vertical bar throughout all the files. The same for any other strayed character. The "not" sign might have been converted to a circumflex or "up arrow". Change it back to the "not" sign, or the tilde if the "not" sign is not avaiable.

2. In lex/init, find the statement beginning with &STLIMIT, and replace it with

```
&STLIMIT  =  -1
```

Also insert two statements:

```
&MAXLNGTH = 4000
NONUM = 1
```

Also in that file remove the input and output association statements, two lines that begin with INPUT('INFILE','INFILE').

Find the statement labeled PRINTPAGE, and replace it with two statements:

```
PRINTPAGE   EJECT(6)
            OUTFILE = L                    :(RETURN)
```

In the last statement of the STOP function a few lines further down (just before the END1.1 label), detach the :(END) and insert the statement

```
        ENDFILE(6)  ENDFILE(7)      :(END)
```

3. In the file lex/core, insert these statements at the beginning:

```
*  I/O associations for UNIX:
     MESSFILE = IDENT(MESSFILE) 'runlog'
     IN       = IDENT(IN)       'tempin'
     OUT      = IDENT(OUT)      'tempout'
     INPUT('INFILE',5,IN)    OUTPUT('OUTFILE',6,OUT)
     OUTPUT('MESSAGE',7,MESSFILE)
```

This allows the files to be overridden in the SPECS file if necesary. Note that these default filenames, e.g. runlog, tempin, may have to be written as full pathnames, e.g. /ab/poto/runlog, /ab/poto/tempin, etc. The interpreter sometimes finds a simple unqualified name as "inappropriate".

4. In the file lex/core, in SEGMENT 4, remove the statement DEFINE('RECONTROL...'). This function is used only by the control module ONBANDS. Use ONBANDSR instead of ONBANDS.

5. In the file lex/sortio find the statement REWIND('INFILE') and change it to REWIND('5'), and in the statement labeled WRITESORTCONTROL, delete the OUTPUT association, i.e., everything but the label.

6. Make two shell scripts, birdbath and toast:

```
#  This is birdbath script, for calling BIRDBATH control modules.
#  To use,
#     1. edit name(s) of dictionary file(s) into first cat
#  statement below.  E.g., if files are dicta and dictb, the
#  statement would read:
#cat dicta dictb > tempin
#     2. Select control module to be used and edit its name into
#  the second cat statement.  E.g., if using LISTGEN, the
#  statement would read:
#cat lex/init specs lex/core lexb/listgen > tempsrc
#     3. Edit a file called specs, placing in it any SPECS
#  statements required by the run.
#     4. Execute this script, assuming it is stored in a file
# called birdbath,  by giving the command
#csh birdbath
#     5. After the run the output should be in the file tempout.
# If it is meant for printing out, send it to the line printer with
# the command
#lpr tempout
# If it is in p-format, and thus destined for editing or for input
# to other programs, it should be saved to another file, e.g. with
```

```
# the command
#mv tempout dicta1.
#     6.  This script normally prints out the program listing and
# run log for diagnostic purposes, and deletes all the temporary
# files except tempout.
#
cat  <input file(s)>  > tempin
cat  lex/init  specs  lex/core  lexa/<control module>  >  tempsrc
spitbol -a  <  tempsrc  >  spitmon
cat spitmon runlog | lpr
rm tempin
rm tempsrc
rm runlog
rm spitmon
#          end of birdbath script


# This is toast script, for calling TOAST control modules.  Set
# it up like the birdbath script (see comments in that file).
# E.g., if running a BANDSORT on dicta and dictb, the first two
# commands below would look like this:
#cat dicta dictb > tempin
#cat lex/init specs lex/core lex/sortio lext/bandsort > tempsrc
# After editing this script, assuming it is stored in a file 'toast',
# execute it by giving the command
#csh toast
# The outputs will be as described in the comments in the birdbath
# script.
#
cat  <input file(s)>  > tempin
cat  lex/init specs lex/core lex/sortio lext/<control module> > tempsrc
spitbol -a  <  tempsrc  >  spitmon
mv runlog runlog1
sort  <  tempout  >  tempin
spitbol -c -x  <  tempsrc  >>  spitmon
cat spitmon runlog1 runlog | lpr
rm tempin
rm tempsrc
rm runlog1
rm runlog
rm spitmon
#            end of toast script
```

7. To execute a birdbath or toast run, follow the instructions in the corresponding script.

8. This is only an elementary way to set up the programs for running. More convenient ways can be devised in which the scripts would not have to be edited but would instead have parameters passed to them when they are called.

APPENDIX I. USING INVERT FOR MAKING FINDERLISTS

This TOAST control module generates a finderlist from a dictionary file. It is called INVERT since the resulting finderlist is in a sense an inversion of the original dictionary. A second control module, INVERT2, is an extension of INVERT with additional facilities described in paragraph r, toward the end of this appendix. Summaries of these control modules are given in Section 3.5 (Chapter 3, Section 5), paragraphs c and d.

This appendix gives in detail the conventions you can use in writing dictionary entries from which a finderlist is to be generated by INVERT. It also describes how to tell the control module what conventions you are using. It assumes familiarity with the elementary notions of band format dictionaries (see, e.g., Chapter 2) and with the general conventions for setting up a TOAST job (see Sections 3.1, 4.1). Section 5.4 Generating Indexes and Finderlists, gives an overview of making and using finderlists.

a. Keywords

Basically, INVERT takes selected keywords (words marked with asterisks) from the definition bands of the dictionary and makes entries in the finderlist with them. From the three dictionary entries in Figure I.1., for example, the program would generate finderlist entries in Figure I.2.

```
.hw   dipwdipw

        df     *grass; *weeds

.hw   mo

        df     *grass; *mulch

.hw   kajau

        df     *omen
```

Figure I.1

```
grass::   dipwdipw,  mo.

mulch::   mo.

omen::    kajau.

weeds::   dipwdipw.
```

Figure I.2

In reading this appendix it will be helpful to distinguish three classes of information being presented here: 1) how you should enter the definition bands in the dictionary; 2) what you need to tell INVERT through SPECS statements (placed in the SPECS file) about the conventions you are using, and 3) what the program does.

Thus, the example in Figures I.1 and I.2 illustrates the following: 1) You have entered definitions in a band labeled df. You have marked each keyword with an immediately preceding asterisk and have separated multiple keywords within a band with a semicolon. The end of the definition band need not be closed with a semicolon. Note that not every entry need have a definition band, nor does every definition band need to have a starred word. 2) To run the program you need to tell the program in what band(s) to look for the starred keywords. For instance, if the band is df (as in the example) use the following statement in the SPECS file,

```
BANDS = 'DF'
```

Note that in the SPECS statement, band names may be given in either upper or lower case--the case need not agree with that in the dictionary file. If the program is to look in both DF and SEM bands, specify both bands:

```
BANDS = 'DF,SEM'
```

In this way, you may specify any number of different bands to be searched for keywords. 3) The program will extract, or copy, each starred keyword (excluding the star) from the specified bands, together with the corresponding headword of the entry or subentry from the headword band, whatever that band may be named--the one or more dots preceding the band name labels the band as a headword band. It will then make up entries for the finderlist, associating each keyword with the corresponding headword. It will arrange the keywords in alphabetical order, and print the corresponding headwords to the right of each keyword, after a double colon. If there are two or more headwords associated with the same keyword, as for "grass" in the example, they are collapsed into one keyword entry, and are given in alphabetical order separated by commas. If a dictionary entry has more than one occurrence of the specified band(s) (at whatever mode levels), the program will look in all of them for starred words. Note that for starred words in a subentry the program associates them with the headword of the subentry, not with the headword of the main entry under which the subentry appears. INVERT2, however, does bring along a reference to the main headword (see paragraph r, below).

b. Phrases

Definitions are rarely single words like "grass" or "mulch", but are more often longer phrases. The program will extract an entire phrase from a definition and place it in the finderlist if there are one or more starred keywords in the phrase. The phrase will appear in the finderlist under each of the keywords. From the dictionary entry in Figure I.3., for example, the program would generate the finderlist entries in Figure I.4.

```
.hw  pas

       df   to *fit *tight
```

Figure I.3

```
fit

      to fit tight::  pas

tight

      to fit tight::  pas
```

Figure I.4

Note that the program deletes the asterisks from the phrases in the finderlist as well as from the keywords before printing out the finderlist. Of course, they remain in the dictionary.

In addition to the entry for "pas" (Figure I.3) in the dictionary, let us add the entries in Figure I.5.

```
.hw  deng

       df   *tight

.hw  jakon

       df   to *fit poorly
```

Figure I.5

The finderlist entries generated from the augmented set of dictionary entries would appear as in Figure I.6 below:

```
fit

        to fit poorly::  jakon.

        to fit tight::   pas.

tight:: deng

        to fit tight::   pas.
```

Figure I.6

Note that the phrases under each keyword in the finderlist are arranged alphabetically ("to fit poorly" coming before "to fit tight").

c. Terminating_the_phrase

If you want to have two or more separate phrases in the definition, that is, you want to prevent the entire definition from being extracted as a single phrase by the program, separate the phrases with semicolons. Let us suppose we have the dictionary entry in Figure I.7.

```
.hw  pas

        df   to *fit *tight;  to *cling
```

Figure I.7

The finderlist program would generate the entries in Figure I.8 below.

```
cling

        to cling::  pas.

fit

        to fit tight::  pas.

tight

        to fit tight::  pas.
```

Figure I.8

The program would not generate the entries in Figure I.9 below.

```
cling

        to fit tight;  to cling::  pas.

fit

        to fit tight;  to cling::  pas.

tight

        to fit tight;  to cling::  pas.
```

Figure I.9

There need not be a semicolon at the end of the band. The end of the band is automatically taken as the end of a phrase.

The semicolon (as well as the end of the band) is the normal signal for terminating phrases. If you want the control module to recognize some other character(s) as phrase terminator(s), you may tell it by inserting in the SPECS file a specification statement of the form:

```
ENDEF = '):;'
```

This example specification statement tells the program to recognize right parenthesis and colon, as well as semicolon, as phrase terminators. Note that only single characters (such as these three characters), and not combinations of two or more characters, can be specified as phrase terminators.

d. Terminating the keyword

We have been talking about keywords as simply "words" that are preceded by asterisks. We now have to be more precise about how the end of a keyword is recognized by the program. As far as the program is concerned, a keyword is terminated on the right by the end of the phrase or by one of the following characters, whichever comes first:

blank, right parenthesis, colon, question mark, exclamation mark, comma, and right angle bracket.

Note that the period is not in this set. To illustrate the effect of keyword-terminating characters, assume we have the following definitions:

```
kind of *fish:   *ray-fish  (*manta);

*when?,  *where?
```

From these, the program would extract the following keywords:

fish

manta

ray-fish

when

where

Note that the question marks are not considered part of the keywords "when" and "where".

These keyword-terminating characters, in addition to the end of the phrase, seem to be what would normally terminate a "word" in the usual sense of the word. You may, however, tell the program to recognize a different set of characters as keyword terminators by putting in the SPECS file a specification statement of the following form:

XENKEY = ', ?!'

In addition to these keyword terminators (whether specified by the program or by the user), another set of characters, normally consisting only of the vertical bar, |, is available for forcing the termination of a keyword before the normal end of the word. If, for instance, the word "gratefully" appears in a definition but you wish only "grateful" to be extracted as the keyword, you would enter the word as

*grateful|ly

into the definition. Then the program would extract only "grateful" as the keyword, and extract "gratefully", including the longer phrase, if any, in which it is embedded, as the phrase. A character which forces the termination of keywords is especially useful for excluding suffixes.

If you wish the program to recognize some other set of characters for forcing the termination of keywords, you may specify them in a specification statement of the following form:

FORCENKEY = '%|'

e. Working copy vs. publication copy

At this point it is convenient to introduce some of the overall motivation for these finderlist extraction conventions. These conventions have been developed over several years by lexicographers and programmers as a compromise between sometimes conflicting requirements. On the one hand, it would be desirable to be able to generate a finderlist mechanically from definitions in a dictionary without any special editorial intervention. On the other hand,

since the choice of material from the definitions for compiling the finderlist cannot be left entirely up to the computer program, some editorial work on the part of the lexicographers has inevitably had to be allowed, subject however to the following limitations: 1) The amount of additional material (special marks, etc.) to be introduced into the definition beyond what would normally have been present should be kept to a minimum, and readability of the copy should be minimally degraded by the added material. 2) No extensive re-wording of the definition should be required. 3) The 'doctored' text (with the additional special marks, etc.) should be readily restorable to its "normal" form by a prepublication editing program. 4) Finally, it should not be too expensive for the finderlist generating program to process these conventions. (In practice this last has been the least heeded requirement.)

Thus, for instance, we have taken as much advantage as possible of the natural punctuation in definitions in delimiting keywords and phrases. In some cases the natural punctuation is not sufficent, such as when the user wishes to terminate a keyword before the normal end of a word. In this particular case, the choice of "|" as the special symbol to force termination of a keyword is motivated both by the unobtrusive and mnemonic shape of the character and by the fact that it need not be "laundered out" at all by the prepublication editing program, since it is the symbol for shifting into roman face, the normal face for definitions. The symbol itself will be suppressed by the photocomposition process.

Therefore, it is desirable to keep in mind that the way text appears in the working printout is not the way it will look in a publication version and to keep in mind exactly what the differences are. So far, in addition to the disappearance of the "|", the main difference is that the asterisks will be suppressed in the publication version. We will have occasion to mention other differences as further conventions are introduced.

f. Using_other_symbols_to_mark_keywords

You may use other symbols to mark keywords if you need the asterisk for another purpose in definitions bands. You must tell the control module what symbol you are using by including (in the SPECS file) a statement such as one of the following:

```
KEYMARK = '%'

KEYMARK = '*
```

The latter statement tells the program that keywords are marked by '*' or '. One use of having two different keyword markers is to mark different kinds of keywords, say one kind for an abridged finderlist, the other set consisting of the additional words that should appear in a full finderlist. To run the abridged finderlist one would specify only the first keyword marker; to run the full finderlist one would specify both markers.

To simplify the prose in the following discussion, however, we will continue to assume that the following statement is true:

KEYMARK = '*'

g. Extending_a_keyword

You may wish the program to extract certain sequences of two or more words as single keywords, as "sea urchin" in the definition

a kind of large sea urchin

If you use the symbol ¬ (hook) (which appears as a tilde on some keyboards) instead of the space between the two words, e.g.

a kind of large *sea¬urchin

the two words will be extracted as a single keyword as in Figure I.10 below.

Figure I.10

The program replaces the hooks by blanks in the finderlist. They remain untouched in the dictionary itself but they can also be replaced there by blanks before publication.

If you wish to use a different character to "hook" words together, override the HOOK parameter, e.g.

HOOK = '='

Do not override it with the null string, that is, do not specify

HOOK =

h. Disambiguating_keywords

If you want the program to generate separate keyword entries for two words that are spelled identically, e.g., "bow" (part of boat) and "bow" (the verb), you may attach different subscripts to the keywords in the definitions, using $ signs, in the same way that headwords in the dictionary are disambiguated. You should, of course, not leave a space between the word and the subscript -- either place the $ right after the end of the word or use a ¬ (hook): bow$2 or bow¬$2. For example, note the entries in Figure I.11.

```
.hw   jimw

        df  part of *bow of canoe

.hw   ruk

        df  to *bow
```

Figure I.11

The entries in Figure I.11 would produce a single keyword in the finderlist as shown in Figure I.12 below.

```
bow

        part of bow of canoe::  jimw

        to bow::  ruk
```

Figure I.12

The situation can be very disconcerting when there are many headwords, answering to the different senses of the one homographous keyword, interleaved with each other (since the phrases are arranged by alphabetical order under a keyword) under the one keyword. Subscripts on keywords can be used to correct the situation, as in figure I.13 below.

```
.hw   jimw

        df  part of *bow¬$1 of canoe

.hw   ruk

       df  to *bow¬$2
```

Figure I.13

These entries would produce two separate keywords in the finderlist as in Figure I.14 below.

```
bow

         part of bow of canoe::  jimw

bow

         to bow::   ruk.
```

Figure I.14

Note that since the subscripts do not appear in the finderlist, one member of a set of homographs need not be labeled with a subscript in the dictionary, if the others are. For instance, the ¬$2 may equally well have been left out in the above example. The program would still have been able to keep the two "bow"'s separate.

These subscripts can, again, be automatically laundered out of the dictionary before publication.

i. Internally_starred_keywords

If you wish a keyword to begin within a typographical word, you may put the asterisk at the desired starting place within the word. For instance, if you wish the keyword "grateful" to be extracted from the word "ungrateful", you would place the asterisk within the word:

un*grateful

If this were the entire definition, the program would generate a finderlist entry like this:

```
grateful
                ungrateful::  ...
```

Figure I.15

This, incidentally, illustrates the fact that when an asterisk is laundered out (either in the finderlist or in the dictionary) it is actually deleted, not replaced by a blank -- otherwise the word in Figure I.15 would become "un grateful". The other side of this convention is that you should be careful to leave a space before an asterisk when it is preceded by a separate word. You should, for instance, write the following in a definition:

to *hit

and not this:

to*hit

A word may have more than one asterisk if you want more than one keyword to be extracted it. For instance, from the following word:

*un*grateful

the program would generate keyword entries for both "ungrateful" and "grateful".

Finally, the forced keyword termination character(s) (see paragraph d) may, of course, be used with internally starred words, to delimit a keyword that is entirely internal to a typographical word as in the following example:

```
un*grateful|ly
```

However, if there is more than one asterisk in a word, the | serves to terminate only the first keyword. It is "used up" by the first asterisk so that the keyword signalled by the second asterisk will not terminate at the |. Thus *un*grateful|ly causes the keywords "ungrateful" and "gratefully" to be generated. If the second keyword is to be terminated in the same place, then two termination characters must be put there. Thus, the definition below would cause "ungrateful" and "grateful" to be generated.

```
*un*grateful||ly
```

j. "Invisible" keywords

If, with all these devices, it is still not possible to isolate the desired keyword in a given phrase because the required sequence of letters does not appear in the phrase, you may place the desired keyword within angle brackets anywhere within the phrase:

```
<*tooth>  baby teeth
```

The definition above would cause the phrase "baby teeth" to appear under the keyword "tooth":

```
tooth

          baby teeth::  ...
```

Figure I.16

Note that angle brackets may contain only keywords, not phrases. There may be any number of starred keywords within the brackets, and keyword-termination within the brackets follows the same conventions as for keywords outside brackets. The difference is that angle brackets with their contents are normally laundered out of the phrase in the finderlist. Hence anything more than keywords in brackets would serve no purpose. Brackets can, of course, also be laundered out of the definition in the dictionary itself before publication.

k. Suppression of laundering

We have noted that many of the special symbols that you introduce into the definitions to cause the desired keywords and phrases to be extracted are automatically "laundered out" (deleted or replaced by blanks) in the finderlist by the program. For some

purposes, you may wish to see the symbols retained in the finderlist. Following are two specification statements that you can place in the SPECS file to instruct the control module to preserve certain symbols: To keep all the asterisks, specify:

```
KEEPSTAR = 1
```

To keep all the angle brackets, specify:

```
KEEPANGLE = 1
```

1. Capitalization of keywords and phrases

In the finderlist, a capitalized keyword is treated as distinct from, and is alphabetized after, the corresponding uncapitalized keyword. Thus, from the two definitions below, the program would produce separate keyword entries, as shown in Figure I.17:

```
*star chart

the North *Star
```

```
star
        star chart::  ...
Star
        the North Star::  ...
```

Figure I.17

There are pairs of words, such as "China" and "china", for which such a separation may be desirable, but for pairs such as "Star" and "star", where listing under the single keyword "star" might be desired, one would have to resort to the angle bracket convention in the definition:

```
<*star> the North Star
```

Phrases, under the same keyword, that differ only by an initial capital vs. lower-case letter will also be treated separately in the finderlist entry. Thus, it is recommended that you not begin definition bands automatically with a capital letter, otherwise two occurrences of the same phrase that happen to differ only in that one occurs at the beginning of the band and the other medially would be treated as different phrases in the finderlist, e.g.

```
fish

          kind of fish::  ...

          Kind of fish::  ...
```

If, for publication purposes the definitions in the dictionary should be capitalized, then the capitalization can easily be performed by a preprocessing program in preparing a publication version. Of course, normally capitalized words should be entered capitalized regardless of position in the band.

m. Phrases and normal punctuation

The INVERT control module, unless instructed otherwise by a specification statement, recognizes the semicolon as the end of a phrase to be extracted. As a general rule, the use of this symbol to terminate phrases to be extracted seems to cause the least dislocation to the normal punctuation of the definition. We specifically have avoided using the period as a phrase delimiter because it is so often used in abbreviations. The user may, as described in paragraph c, specify other characters as phrase terminators.

In individual definitions, the user may force the termination of a phrase, for extraction purposes, at a particular punctuation mark by placing a phrase terminator (the semicolon, unless otherwise specified) either before or after that mark. Dictionary definitions containing such a phrase might look like these:

```
*how many?; how much?

to *hit;, to *beat
```

The phrases extracted from the definitions above would be

```
how many?

to hit

to beat
```

Note that the question mark is part of the first phrase. The comma is not part of the second phrase. Even though the comma should be part of the third phrase, because it comes after the semicolon that terminates the previous phrase it has been laundered out by the program. A particular set of characters, when occurring at the beginning of a phrase, as this comma does, are automatically deleted by the finderlist program. The set is given below:

```
,.;:?!
```

You may respecify this set with a specification statement in the SPECS file as shown below:

```
OPTP = '.?!'
```

In preparing the dictionary for publication, compound punctuation such as ?; and ;, can be reduced to their normal form by a

preprocessing program. The two dictionary definitions in the example above would look like this after such processing:

how many? how much?

to hit, to beat

n. Keywords and face shifts

When you mark an italicized word as a keyword you have the choice of placing the asterisk before or after the italic shift character, %. If you place it before, as in *%land ho! the keyword extracted will include the font shift. If placed after, as in %*land ho!, the keyword will not carry the italic shift character in the finderlist, even though the phrase will, as in the example below:

```
land

        %land ho!::  ...
```

Figure I.18

o. Deletion of initial keyword

This feature has not been implemented in the current version of INVERT.

This feature, called PHRASEDIT, if activated by the user through a specification statement, would delete in each phrase in the finderlist a phrase-initial keyword that is immediately followed by a colon and space. This would be useful if the dictionary contains many definitions in the style shown in the following examples:

*fish-species: *sergeant-major

*fish-species: large *skipjack

*fish-species: *yellow-spotted-*trigger-fish

With the PHRASEDIT feature inactive, the program would generate from the above definitions the following finderlist entries under fish "species":

```
fish species
        fish species:  large skipjack::  ...
        fish species:  sergeant major::  ...
        fish species:  yellow-spotted trigger fish::  ...
```

Figure I.19

Such redundancy would be suppressed by activating the PHRASEDIT feature, resulting in the following entry:

```
fish species
        large skipjack::  ...
        sergeant major::  ...
        yellow-spotted trigger fish::  ...
```

Figure I.20

The specification statement to activate the feature is as follows:

PHRASEDIT = 1

This feature is not available in the current version of INVERT and of INVERT2.

p. Format of the finderlist listing

The finderlist is itself in band-format. It is a dictionary with two bands, KW (for keyword) and PH (for phrase). Since it is in band-format, it can be printed out in 1-format by the LISTGEN control module. A typical entry in 1-format, complete with band labels, might look like this:

```
51   .kw   afraid::  loamw,  majpwehk,  mijik
52              ph  afraid (of physical harm)::  kajanjan.
53              ph  afraid of ghosts::  lehngin.
54              ph  afraid, worried::  per.
```

Figure I.21

Each entry has a KW band, and zero or more PH bands, depending on how many different phrases contain the given keyword. There are no modes or subentries. In all the previous examples in this Appendix, the band labels have been suppressed; this is one of the options available with 1-format printouts, and it is the normal one taken with 1-format output from INVERT, since with only two different bands, one always indented and the other always at the left margin, showing their names would be superfluous.

In general, the full range of format options available to 1-format listings are available for finderlists. However, since there are no modes or subentries, the options boil down to those in the following list. (Of course, if you are not generating a listing but only a p-format file, not even these are relevant.) The default value is shown with each parameter. Since these specification statements pertain to Phase 2 of the INVERT control module, they must be placed in the phase-2 SPECS file, not in the phase-1 SPECS file.

LMARGIN = 0 width, in number of spaces, of left margin

NONUM = adds a serial number field. Note that each band normally is numbered, as in Figure I.21. To suppress this field, set the parameter to 1: NONUM = 1 .

PRINTBANDLABELS = The old name for this parameter was KEEPBANDLABEL, which still works, but will be superseded by PRINTBANDLABELS. The value is normally null, causing bandlabels in the finderlist not to be printed. Set it to a non-null value, e.g., PRINTBANDLABELS = 1 to cause the bandlabels to appear.

RMARGIN = 132 the right margin

LINESPERPAGE = 60 the maximum number of lines (including blank lines) per page. For 11-inch deep paper printed at 8 lines per inch, use LINESPERPAGE = 80 .

HEADHANG = 10 the number of spaces the keyword band overhangs the following bands in an entry.

Finally, you may provide a title to be printed at the top of each page of the finderlist if you are requesting a listing. This is provided in a statement:

TITLE = 'SAMPLE FINDERLIST'

The Trussel Feature causes the position of the phrase and headword to be interchanged, i.e., the headword now appears to the left of the phrase, and phrases are in order by headword. This feature is activated by the specification statement:

```
TRUSSEL = 1
```

which must appear in both the phase-1 SPECS file and the phase-2 SPECS file. With this feature turned on, the example in Fig. I.6 would appear as in Fig. I.22:

```
fit

          jakon::  to fit poorly

          pas::  to fit tight

tight:: deng

          pas::  to fit tight
```

Figure I.22

g. Alphabetical ordering of keywords

Keywords are normally alphabetized with the usual English alphabetization conventions, which ignore the distinction between upper and lower case, and ignore spaces, hyphens, and other non-alphabetic symbols. The program does this by using a "handle" function, called 'HANDLEB' built into the TOAST system, which generates from each word a new form called a "handle" by converting all lower case letters to caps and dropping all non-alphabetic symbols. If a different handle is desired, a different function must be written and placed in the phase-1 USERFNS file for the job (see, e.g., recipe in Section 4.3, Fig. 4.3.6.) The user may define this function as 'HANDLEB'. Alternatively, a different name may be used, say 'ENGHAND', but then a statement alerting the program to the different name must be placed in the phase-1 SPECS file, of the form,

```
HANDLEBNAME  =  'ENGHAND'
```

For alphabetizing the source language forms in finderlist entries, e.g. the three forms loamw, majpwehk, mijik in Fig. I.21, a different built-in handle function is normally used, called 'HANDLE'. This function does the same things as the built-in HANDLEB function. The user may, again, supply, in the phase-1 USERFNS file, a handle function more appropriate for the language, and call it either 'HANDLE' or something else, e.g., 'MOKHANDLE'. If the name is not 'HANDLE', then a statement must be placed in the phase-1 SPECS file to alert the program to the unexpected name, e.g.

```
HANDLENAME  =  'MOKHANDLE'
```

r. Subentries and translation bands

When keywords appear in the definitions of subentries, INVERT brings the headword of the subentry into the finderlist, not the headword of the dominating main entry. That is, the finderlist does not show the reader where, i.e., under what main headword, to locate the subentry. INVERT2, an expanded version of INVERT, does bring along both the main headword and the subheadword. Furthermore, stars may be placed in "translation" bands for illustration bands. Details follow.

1. If a definition is in a subentry, not only the headword of the subentry is copied into the finderlist entry, but also the main headword. If the definition is in a sub-sub or lower entry, then the corresponding sub-sub (etc.) headword is copied, but not the intermediate level headwords.

2. Bands containing examples or illustrative expressions can be treated as sub-headwords for purposes of generating a finderlist if each such band is paired with a translation band with the same mode:

```
.hw     headword, main or sub-.
          df    definition of headword
            1ex     example
            1so     possible intervening band(s)
            1tr     translation of *example
```

In the above example, the tr band can be treated as the definition band corresponding to the "subheadword" in the ex band. Note that these two bands have to be in the same mode, that the ex band has to precede the tr band, and that there may be intervening bands (which must be in the same mode). The example bands must be specified in a phase-1 SPECS statement assigning a list of their names to XBANDS , and the translation bands must be specified with a statement assigning their names to TBANDS, e.g.

```
XBANDS  =  'EX,EXP'
TBANDS  =  'TR'
```

In the dictionary file it is better to use different band names for a band giving a definition for the headword and a band giving the translation for an example or expression, although the program will assume, if they have the same name, that such a band having the same mode as an XBAND is a translation band rather than a definition band.

If the band name of the subheadword band or of the example band is to be brought along into the finderlist entry, then put

```
INCLBANDNAME  =  1
```

in the phase-1 SPECS file.

APPENDIX III. COMPUTERIZED LEXICOGRAPHY AT UH

COMPUTER-AIDED COMPILATION OF NATURAL-LANGUAGE DICTIONARIES AT THE UNIVERSITY OF HAWAII: A SKETCH

May, 1983

1. Introduction

This is a minor update of a memo of the same title that first appeared in January, 1975. It is included here in order to give a little background on the environment in which the programs were developed.

2. The Dictionaries

Since 1965, dictionaries for some 40 languages of the Pacific basin have been computer-processed here, and one or two new ones are begun each year. At this writing about twenty have been published or are in press and several are nearing completion. A number of lexicons that are no more than computerized wordlists are not included in this count. A list of the dictionaries appears at the end of this appendix. The first dictionary to have been processed by computer here was the Maranao Dictionary by Howard McKaughan.

About half of the languages concerned are languages of Micronesia. The rest are languages of the Philippines and of other areas in the Western Pacific basin, with the exception of several Amerindian languages of the Pacific Northwest.

The dictionaries are bilingual, with English being the language used in the body of the entries. Each of the dictionaries is the first extensive one compiled so far for the particular language. Each language has had little or no previous written materials. The dictionaries are usually intended for a heterogeneous audience consisting of vernacular speakers learning English as a second language, English-speakers learning the vernacular language, and other linguists.

Those that have been completed or are nearing completion typically contain five or six thousand entries, with some having as many as ten to twelve thousand. Each entry typically includes, besides English definitions, such information as the part-of-speech and other grammatical information, derivational privileges, loan source where appropriate, dialect and other variant forms, scientific identification of flora and fauna, and example sentences with translations. Some of the dictionaries further include synonyms and antonyms, morphological analysis of the words where appropriate, idioms, derived and inflected forms where these are not readily predictable, reconstructions in proto-languages, phonemic transcription if the headword is given in some non-phonemic orthography, etc.

Typically each dictionary has been developed by one linguist, who often has also produced (or is concurrently producing) a grammar of the language in collaboration with one or more vernacular speakers. In the case of the Micronesian languages, work on standardization of the orthographies has also been concurrently undertaken. Data for each dictionary has been gathered both in the field and from informants brought to Hawaii. In several cases the linguists themselves have been based at other universities, at least for part of the duration of their projects. So far all the data entry and computer processing has been done here. The bulk of project and computer funding has come from extramural grants from various agencies. We have also received a substantial amount of in-house support, especially for computer time for systems development and for dictionary projects not yet externally funded.

Some of the authors had been compiling their dictionaries for years (in some cases up to twenty years) before turning to the computer for further expansion and refinement. Others started to use the computer from the beginning of compilation.

3. Practical Operation

After an initial batch of entries (which may be anything from a bare wordlist with minimal definitions to several shoeboxfuls of slips gathered over many years) has been keypunched or entered into the computer and printed out, the typical sequence of events in further processing has been as follows. The linguist would work over his printout with an informant, checking and expanding the information in entries and adding new entries. He would write corrections and additions directly on the printout, which would later be used to revise the computer file. (In the early days, the linguist revised through keypunched edit instructions.) A new printout of the revised file would then be produced. This is the basic cycle, to which other operations, such as producing other printouts and running programmed revisions to the file, are frequently adjoined. Some of these other types of operations are described below. During each cycle the linguist may choose to concentrate on a limited number of tasks, such as refining the definitions, eliciting derived forms, adding example sentences, checking the transcriptions, etc. A cycle would last anywhere from a few weeks to a year or more. Typically a dictionary ready for publication would have been through more than ten of these cycles, with the last four or five devoted primarily to matters of accuracy and consistency rather than to the addition of new material.

4. Advantages of Computer Processing

The demands put on the computer by the lexicographers here have been consistently of the data-processing and data-organization variety. The functions made possible by computerization that are most frequently in demand are: 1) editing of the dictionary file and sorting in of new entries, 2) generating finderlists, and 3) printing of duplicate copies for sending to workers in the field.

Some of the other functions we frequently perform are the following: 4) retrieval of specified subsets of entries (e.g., botanical terms), 5) producing phoneme concordances of headwords, 6) generating derived forms and cross-reference entries, 7) checking for consistency, 8) sorting by different alphabetization schemes, 9) systematic conversion of orthography throughout a dictionary, and 10) automatic insertion of typesetting control symbols for computerized photocomposition. Most of the dictionaries published or about to be published have been or will be typeset by computer-controled photocomposing systems.

In the case of the Northwest Amerindian languages, we have made a continuous effort to find devices that would display data using the phonetic symbols to which linguists working with those languages are accustomed. When the time-sharing system first became available we were able to use a Selectric terminal (the IBM 2741) to enter, edit, and type out the dictionary materials. The terminal was equipped with a Selectric type element that had been designed specially for this family of languages. Due to the heavy reliance on phonetic symbols and diacritics in the transcription of these languages, we would not have undertaken computer processing of these dictionaries if we had had only the limited fonts of keypunch and standard print trains. Later we acquired some phonetic symbols for the print train on the high-speed line printer at the computing center for large volume print-outs. More recently we have acquired microcomputers (made by Terak) which allow the user to design characters for display on the screen, and also a Sanders Media 12/7 dot matrix printer equipped with special character fonts.

The aspect of computer processing of dictionaries in which there has been most constant demand is the automatic generation of finderlists. A finderlist is an index of selected words occurring in the definitions, each generally accompanied by some portion (a phrase) of the definition. A large number of conventions have grown up in connection with punctuation and other special marks to be used in the definitions for specifying what words and phrases are to be copied into the finderlist. The purpose of this sytem is to enable a usable finderlist to be automatically generated from material already in the definitions, with a minimum of additional intervention from the author. Apart from providing a further tool for elicitation and ultimately a useful adjunct to the published dictionary, a finderlist turns out to be valuable in bringing out inconsistencies and errors of phrasing, spelling, and punctuation, etc., in the dictionary.

Another tool that we have found to be extremely useful for a wide variety of purposes (from checking for errors and inconsistency to indexing on categories of information in the dictionary entries) is simply a closely spaced alphabetical listing of the contents of specified fields, accompanied by the relevant headwords.

Besides programs we have, of course, had to develop suitable data formats. In this area also, user demands have been the principal

guide. Our current data format reflects a rather general conception of the structure of dictionary entries. It allows a lexicographer to assign his own names to the fields in his entries, allows fields of unlimited length, and allows repetition of fields and arbitrary nesting of fields. A facility for subentries and sub-subentries to any depth (for derived words, etc.) provides another dimension of nesting. This format so far has been most welcome as a mold to help the lexicographer organize his information. The printing program displays this organization clearly in the format of the printout.

5. Machinery

The hardware that has been available to us has consisted (except in the very early days) of machines of the IBM S/360 and 370 family. A time-sharing system, TSO, has in addition been available on these machines since about 1970. However, due to limitations of disk space and other hardware parameters, we did not rely heavily on TSO for on-line editing and other work until quite recently. While the editing of dictionary files is now largely done on-line, the dictionary programs themselves are still run in batch mode, which is more economical. Currently some dictionary editing is done on microcomputers. The files are then transmitted to the mainframe for batch processing.

Programs were originally written in Fortran and later in PL/1. When Spitbol, the fast-running implementation of SNOBOL4, became available on the 360, we gradually switched over to that language. Spitbol has allowed us to experiment with, and implement, features easily (almost casually) that we would not have considered writing in PL/1 or Fortran. All our current programs are written in Spitbol and are linked together for convenient use by generous amounts of JCL, profiting especially from certain JCL-oriented features of Spitbol. In addition, the efficient Sort/Merge program available under OS has been a mainstay of our operations.

A recent grant from the National Science Foundation has permitted us to consolidate and make more uniform the programs that have grown up over the years. Better documentation is also being written. The goal is to make the programs more easily used and expanded, and more easily transportable to other installations.

Below is a list of the languages for which dictionaries have been processed by these programs or their predecessors. The authors, and publishers (for those dictionaries that have already been published), are also given. "UPH" stands for University Press of Hawaii.

Banoni (Piet Lincoln)
Bontoc (Lawrence Reid) Australian National University 1976
Chamorro (Donald Topping) UPH 1975
Colville (Tony Mattina)
Itawis (Jim Tharp) HRAFlex 1976
Kagayen (Carol Harmon)
Kapingamarangi (Michael Lieber) UPH 1974
Kiribati (Gilbertese) (Steve Trussel)
Klallam (Claudine Poggi)
Kmer (Philip Jenner)
Kusaiean (Kee-Dong Lee) UPH 1976
Maori (Bruce Biggs) (English-Maori:) Auckland University Press 1981
Maranao (Howard McKaughan) UPH 1967
Marshallese (Byron Bender) UPH 1976
Mokilese (Shelly Harrison) UPH 1977
Mortlockese (Martin Combs)
Nakanai (Ward Goodenough)
Nukuoro (Vern Carroll) UPH 1973
Palauan (Lewis Josephs) UPH 1976
Pingilapese (Elaine Good)
Ponapean (Ken Rehg) UPH 1979
Pulo Annian (Sachiko Oda)
Puluwat (Sam Elbert) Australian National University 1972
Rennellese (Sam Elbert) (English-Rennellese) National Museum of Denmark 1981
Rhade (Jim Tharp) Australian National University 1980
Rotuman (Joel Fagan)
Saipan Carolinian (Rick Jackson)
Sora (David Stampe)
Sre (Niall Olsen)
Spokane (Barry Carlson)
Takuu (Irwin Howard)
Tahitian (Jack Ward)
Thompson (Laurence Thompson)
Tillamook (Laurence and Terry Thompson)
Trukese (Ward Goodenough) American Philosophical Society 1980
Tuvaluan (Ellice Islands) (Nicholas Besnier, Steve Trussel) Peace Corp 1981
Woleaian (Ho-Min Sohn) UPH 1976
Yapese (John Jensen) UPH 1977

END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
EN JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG
END JOB 7130 ACCT 1992 T119920S 9.54.54AM 02FEB90 PRT1 =A T=TN C 180 PG