KANSAS EVENT DATA SYSTEM
K - E - D - S

Philip A. Schrodt
Dept. of Political Science
University of Kansas
Blake Hall
Lawrence, KS 66045
785.864.9024 (phone) - 913.864.5700 (fax)
Email: p-schrodt@ukans.edu

VERSION 1.0
February 1998

 Program development funded by the National Science Foundation through Grants SES89-10738, SBR-9410023 and SES90-25130 (Data Development in International Relations Project) and the University of Kansas General Research Fund Grant 3500-X0-0038. Brad Bennett, Doug Bond, Joe Bond, Shannon Davis, Ronald Francisco, Deborah Gerner, Phillip Huxtable, Tony Nownes, Jon Pevehouse, Julia Pitner, Scott Savaiano, Uwe Reising, and Judy Weddle have served the difficult role of testing KEDS.

Updated copies of the KEDS program and manual can be found at the URL
http://www.ukans.edu/~ keds

Please Cite Program As:
Philip A. Schrodt. 1998. KEDS: Kansas Event Data System. Version 1.0.

Contents

LIST OF ACRONYMS

ASCII	American Standard Code for Information Interchange
BCOW	Behavioral Correlates of War data set
COPDAB	Conflict and Peace Data Bank
DDIR	Data Development in International Relations project
GEDS	Global Event Data System (University of Maryland)
ICPSR	Inter-university Consortium for Political and Social Research (Ann Arbor, Michigan)
K	Kilobyte
KEDS	Kansas Event Data System
MB	Megabyte
NEXIS	A data service of Mead Data Central
NSF	National Science Foundation
PANDA	Protocol for the Analysis of Nonviolent Direct Action (Harvard University)
TEXT	The Macintosh "flat ASCII" text format
UN	United Nations
WEIS	World Events Interaction Survey

Chapter 1

INTRODUCTION

The Kansas Event Data System (KEDS) is a system for the machine coding of international event data based on pattern recognition. It is designed to work with short news summaries such as those found in the lead sentences of wire service reports or in chronologies. To date, KEDS has primarily been used to code WEIS events (McClelland 1976) from the Reuters news service but in principle it can be used for other event coding schemes.

Historically, event data have usually been hand-coded by legions of bored undergraduates flipping through copies of the *New York Times*. Machine coding provides two advantages over these traditional methods:

- Coding can be done more *quickly* by machine than by hand; in particular the coding of a large machine-readable data set by a single researcher is feasible;
- Machine coding rules are applied with complete *consistency* and are not subject to inter-coder disparities caused by fatigue, differing interpretations of the coding rules or biases concerning the texts being coded;

The disadvantage of machine coding is that it cannot deal with sentences having a complex syntax and it deals with sentences in isolation rather than in context.

KEDS can be used for either machine-assisted coding or fully automated coding. Coded events can be manually edited on the screen before they are written to a file, and the program has a "complexity detector" that can divert linguistically complex sentences – for example those containing a large number of verbs or subordinate clauses – to a separate file for later human coding.

This documentation assumes a basic familiarity with event data coding and is primarily a guide to the use of the KEDS program rather than as an introduction

to developing an event data coding scheme. The documentation also assumes familiarity with the basic operations of the Macintosh operating system.

1.1 How KEDS Works

KEDS uses a system of pattern recognition to do its coding. Three types of information are used:

Actors: These are proper nouns that identify the political actors recognized by the system;

Verbs: Because event data categories are primarily distinguished by the actions that one actor takes toward another, the verb is usually the most important part of a sentence in determining the event code.

Phrases: Phrases are used to distinguish different meanings of a verb – for example PROMISED TO SEND TROOPS versus PROMISED TO CONSIDER PROPOSAL – and to provide syntactic information on the location of the source and target within the sentence.

KEDS relies on *sparse parsing* of sentences – primarily identifying proper nouns (which may be compound), verbs and direct objects within a verb phrase rather than using full syntactical analysis. As a consequence KEDS will make errors on complex sentences or sentences using unusual grammatical constructions, but it requires less information to deal with the sentence structures most commonly encountered in news articles. By foregoing the use of a full parser, KEDS is quite robust in correctly interpreting the types of English sentences found in newswire reports.

1.2 Advantages of machine coding

We originally became involved with machine coding because it is dramatically faster and less expensive than human coding. Once a researcher has established vocabulary lists of actors and verb phrases, the only cost involved in generating event data is the acquisition of machine-readable news reports. (These are increasingly available from CD-ROM and electronic networks.) Furthermore, a coding system developed at one institution can be used by other researchers through the sharing of vocabulary lists and coding software; this has been part of our collaboration with the PANDA project (see below). The ability to modify the coding system quickly is essential in policy applications, since decision-makers must often deal with situations that may not have been addressed earlier by academic researchers.

In working with KEDS, we discovered two additional advantages to machine coding. First, it is free of non-reproducible coding biases. Human coding is subject to systematic biases because of assumptions made by the coders. For example, Laurance (1990) notes that even expert coders at the U.S. Naval Postgraduate School tended to over-estimate the military capability of China because they knew China to be a large Communist country. Because most human event coding is done part-time by students, coder biases are difficult to control. In contrast, in machine-coding the words describing an activity will receive the same code irrespective of the actors or time period involved. Sparse parsing also tends to make *random* coding errors that a statistical analysis can rectify; the systematic errors introduced by human coders are much more difficult to correct. Any biases embedded in the machine coding system are preserved *explicitly* in its vocabulary; there is no such record in human coding.

Second, it is much easier to experiment with alternative coding rules using machine coding. The COPDAB (Azar 1982) and WEIS (McClelland 1976) event coding schemes are very general and for the most part presuppose a Cold War, Westphalian-Clausewitzian conflict framework. This weakens the value of WEIS and COPDAB data when dealing with post-Cold War phenomena such as ethnic conflict, low-intensity conflict, and multilateral intervention. Using a machine-coding system, even a very large data collection such as our Arab-Israeli conflict data set (100,000 events) can be completely recoded in a couple hours. This is impossible with human coded data, which has severely restricted experimentation with new coding schemes.

1.3 History

The development of KEDS began around 1990 as part of the National Science Foundation's "Data Development in International Relations" project (Merritt, Muncaster and Zinnes 1994). While the DDIR work included some experimentation with German-language sources and foreign policy chronologies (Gerner et al 1994), most of our experimentation at Kansas with the English-language KEDS has been done with 3-digit WEIS codes on interactions reported by Reuters in the Middle East. We are currently maintaining an event data set covering the Levant and Gulf areas of the Middle East that covers 1979 to the present; the data set contains over 100,000 events.

The other major project using the KEDS program is the Protocol for the Assessment of Nonviolent Direct Action (PANDA) at the Program on Nonviolent Sanctions in Conflict and Defense at the Center for International Affairs at Harvard (Bond, Bennett, and Vogele 1994). This project uses KEDS to code a superset of the WEIS categories (160 categories versus the 63 categories in WEIS) that provide far more detail on nonviolent events, substate actors and internal interactions such as strikes and protests. PANDA codes several contex-

tual variables in addition to the standard date-source-event-target variables of event data. Reuters reports dealing with the entire world have been coded for 1985-1994; the resulting data set contains about 500,000 events.

In addition to these large projects, several additional KEDS dictionaries have been developed by graduate students. These include

- Behavioral Correlates of War (BCOW; Leng, 1987) coding system of the Middle East (Jon Pevehouse)
- WEIS coding for West Africa, using full-story coding (Phillip Huxtable)

The version of KEDS described in this manual has almost all of the features that we anticipate will be in KEDS 1.0.

1.4 System Accuracy

The accuracy of KEDS depends heavily on the source text, the event coding scheme and the type of event being coded. We have done a variety of different reliability checks in recent papers and articles (Gerner et al 1992, Schrodt et al 1992, Schrodt and Gerner 1993,1994 , Schrodt 1993). These papers are available on the KEDS web site, or feel free to contact us for copies.

With Reuters lead sentences and the WEIS coding scheme, KEDS's will assign the same code as a single human coder in about 80% to 90% of the cases. Approximately 10% of the Reuters leads have a syntactic structure that is too complicated or too idiosyncratic for KEDS to handle properly, although some of the residual coding disagreement comes from ambiguities in the WEIS coding categories themselves.[1] In an experiment where dictionaries were optimized for the coding of a single day of Reuters leads, the PANDA project – using a coding scheme substantially more detailed than WEIS – achieved a 91.7% machine coding accuracy; this probably represents the upper limit of accuracy for Reuters leads and a program using KEDS's sparse parsing approach (Bond, Bennett & Vogele 1994:9). This level of coding accuracy is comparable to that achieved in event data projects using human coders: Burgess and Lawton (1972:58) report a mean intercoder reliability of 82% for eight projects where that statistic is known.[2]

[1] Examples of sentences that are too complex to code include the following:

`The United States on Friday dismissed Israel's apparent rejection of an Egyptian plan for talks with the Palestinians as 'parliamentary maneuvering' and said the door was not closed to peace.`

`Resumption of ties between Egypt and Syria may spur reconciliation between Iraq and Syria, and Syria and the PLO, the Qatari newspaper` *al-Raya* `said on Friday.`

[2] KEDS had a somewhat lower agreement when compared to multiple human coders. In many cases this was not because KEDS was coding poorly but because KEDS was more

Schrodt and Gerner (1993, 1994) assess the face validity of KEDS-generated data for the Middle East, 1982-1993; the time series produced by the program correspond closely to the patterns expected from narrative accounts of the interactions between the actors. In these papers, the KEDS data were also compared to the human-coded WEIS data set for 1982-1991. For almost all dyads, there was a statistically significant correlation between the number of events reported by the two series, as well as the number of cooperative events. On the net cooperation values aggregated using the Goldstein (1992) scale and number of conflictual events there was a statistically significant correlation in about half of the dyads. Many of the differences between the two series appear to be due to the higher density of events in KEDS compared to the *New York Times*-based WEIS: the Reuters series contained, on average, three times as many events as WEIS. The KEDS and WEIS data sets were also used in two statistical studies - one involving cross-correlation and the other spectral analysis - produced generally comparable results, although some idiosyncratic differences are found in specific dyads.

Other work in automated text processing of reports of political events (ARPA 1993; Linert and Sondheim 1991) indicates that dictionaries on the order of about 5,000 words are necessary for relatively complete discrimination between political events described by news media sources. The dictionaries used in these validity studies were somewhat smaller than this and the accuracy with our current dictionary – about 4000 phrases - may be somewhat higher.

1.5 The KEDS Web Site

The KEDS project maintains a web site at the URL

`http://www.ukans.edu/~ keds`

At this site you will find the most recent versions of the software and this manual, assorted dictionaries, data sets and utility programs, a FAQ (frequently-asked-questions) section, and copies of papers from the project.

1.6 The *KEDS.Sample* Demonstration

If you received a copy of this manual via a disk or set of files containing a Sample folder, that folder contains a set of hypothetical events that demonstrate a number of the features of the program.[3] Start the KEDS program by double-clicking the KEDS program icon. An introductory screen will be displayed,

consistent and less likely to miss multiple events in a single story than some of the human coders (Gerner et al. 1992, Schrodt, et al. 1992).

[3]The Sample *f* folder can be downloaded from the web site.

followed by a dialog box asking for the coder ID. Enter any text (e.g. your initials) in response to this, then click OK.

The next dialog box asks whether you want to use an existing coding file or initialize a new file. Click the **Use Existing Project**

File button: this will present a standard Macintosh file selection dialog. Double-click the **Sample** folder to open it, then double-click the **KEDS.Sample** file.

After reading the dictionary information, the program will begin coding the first record in the file.[4]

The demonstration sentences in **Sample.Text** illustrate a number of features of the KEDS system; use the <Return> key or the arrow in the lower right corner of the screen to advance through the text. The text of the events uses the syntax and vocabulary typical of Reuters newswire lead sentences.

1.7 System Requirements

We have used KEDS in a variety of Macintosh configurations, including an SE, SE/30, original II, II with a DayStar Turbo 30 accelerator, IIsi, IIsi with a DayStar Turbo 40 accelerator, Quadra 900, LC, Powerbook 160, Powerbook 520c, and Power Macintosh 5400 and 7100; we've also used it under Systems 6.0.5, 6.0.7, 7.1, 7.5 and 8.0. The suggested application memory size is set at 2048K so KEDS should run on Macs with 4Mb or more of memory under System 6, or 8Mb under System 7.[5]

KEDS uses the following display fonts: Monaco (9 and 12 pt), Geneva (10 and 12 point) and Helvetica (10, 12, 14 and 24 point; the latter two fonts are used only in the introductory screen).

1.8 Legal Stuff

Under the Bayh-Dole Act governing technology developed with National Science Foundation funding, the KEDS program is the intellectual property of the University of Kansas. You may use and make copies of the program for educational, government and non-profit use without charge; the program can be posted to bulletin boards and included in software collections provided that a

[4] KEDS can also be launched by double-clicking the icon for the KEDS.Sample file: in this case the file selectrion step will be bypassed.

[5] The 2Mb requirement is actually quite conservative in order to allow for memory-intensive operations such as indexing. If you are low on memory, the program will probably run safely with about 1.4Mb of memory; the minimum required memory can be changed in the "Get Info" box of the program.

copy of this manual is included. If you wish to license the program or its source code for commercial applications, please contact the Technology Transfer Office, Office of Research, Graduate Studies and Public Service, Research Support and Grants Administration, University of Kansas, Lawrence, Kansas (phone: 785-864-3302; fax: 785-864-5272).

THE KANSAS EVENT DATA SYSTEM (KEDS) COMES WITH NO WARRANTIES, EXPRESSED OR IMPLIED. Philip A. Schrodt and the University of Kansas do not warrant that the software is free of bugs or omissions, and they make no representations, expressed or implied, with respect to this software, its fitness for a particular purpose, or warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free (quite the contrary...). In addition to the foregoing, you should recognize that all complex software systems and their documentation contain errors and omissions (see below); Philip A. Schrodt and the University of Kansas shall not be responsible under any circumstances for providing information or corrections to errors or omissions discovered at any time in the product, whether or not they are aware of the errors or omissions. Philip A. Schrodt and the University of Kansas does not recommend the use of the software for applications in which errors or omissions could threaten life, injury or significant loss, including the failure to complete research projects.

1.8.1 Because you are undoubtedly wondering...

The term KEDS is an acronym for "Kansas Event Data System." The software is in no way connected to – nor could it possibly be confused with – a trademarked brand of footwear with a similar name.

1.9 Typeface conventions used in this manual

Menu options, dialog button labels and keys are in **Helvetica bold**. The path to an option within a menu is shown using slashes, e.g.

DISPLAY/Status/Project

File types begin with a period and are in italics, e.g. *.Verbs, .Actors, .events*. Input files begin with an upper-case letter; output files with a lower case letter. The output file suffixes are assigned by KEDS; the input file suffixes are the ones we've used in our project.

Chapters and sections of the manual are in Helvetica plain.

Commands are in **bold**

Input text is in `Courier plain`

KEDS output is in SMALL CAPS

◇ represents a blank

1.10 Definitions

Source refers to the actor in an event data record who initiated at action; in some event data discussion this is called the "actor"

Target refers to the actor in an event data record who is the object of an action

Case-sensitive means that the program differentiates between upper- and lower-case letters. In a case-sensitive command, the keyword **SOURCE** will be recognized but **Source** and **source** would not be recognized.

String refers to any set of consecutive characters

Literal means a string that does not contain symbols that are interpreted by KEDS, for example

```
*, $, +, %, |, {, }, ~
```

Token means a string of characters that has a special meaning to KEDS, for example

```
*, $, +, %, <VERB>, <TEXT>, ->
```

x-delimited means a string that is between the characters 'x.' For example, in the sentence

`President Clinton, arriving in Dublin, said that US policy...`

the phrase "`arriving in Dublin`" is comma-delimited. In the class definition

<tobe> ◇=◇IS_◇WAS_◇WERE_◇WILL_BE_◇

the strings "IS_," "WAS_," "WERE_," and "WILL_BE_" are space-delimited.

ASCII stands for American Standard Code for Information Interchange; it is the standard code through which characters are stored in a computer and in files. For example, A is represented by the number 65; a blank is the number 32 and so forth.

Macintosh TEXT file means a file that contains only characters, without additional formatting information. Virtually all word processing programs will read and write a TEXT file; if you are creating a new file, use the word processor's Save As...Text Only option for any files that will be read by KEDS. These files are occasionally called flat ASCII.[6] Macintoshes running System 7.0 can also directly read the TEXT files produced by programs running in more primitive operating systems such as Windows and DOS; Macs running System 6.0 and earlier can do this using the Apple File Transfer program.

Phase refers to a set of words and tokens in the *.Verbs, .Actors, .Classes* or *.Options* files.

Clause refers to an element of a text sentence that is delimited by either conjunctions or commas.[7]

A **default** is the value that a parameter will have unless it is changed by a command or menu option.

1.11 Bugs and Extensions

PLEASE REPORT BUGS! This is beta-ware; it is supposed to have bugs and it will not disappoint you in that regard. Versions since 0.6 seem to be fairly stable and have been used extensively by the PANDA project and several graduate students, but it *may* still contain a bug that periodically destroys the *.Verb* list. Keep extra backups of the *.Verb* and *.Actor* files; don't depend on KEDS's backups alone!

KEDS is an on-going project and the dictionaries, in particular, are continually being updated. If you intend to use the program in an actual research project, check the web site for the latest copy of our dictionaries and the program, and a list of other projects that are willing to share their dictionaries.

> **Note:** Bug reports are absolutely vital in the development of a complex program such as KEDS, so don't be shy! We've only been able to test the program on a small number of hardware configurations and we've done most of our work with the same set of texts and vocabulary lists. If you find a situation where KEDS crashes, or

[6] The files are "flat" because they are not structured with formatting information.

[7] In English grammar, a clause technically is any group of words – a subject and predicate. Allowing for the implied subject in compound sentences of the form

```
Arafat arrived in Jordan today and met with King Hussein
```

most clauses parsed by KEDS probably satisfy this criterion – at least in Reuters leads – as do all clauses coded by KEDS, but the markers used to delineate the clauses are commas and conjunctions.

does not behave as described in the manual, please let us know – this might be symptomatic of a critical problem in the code or the documentation.

1.12 Suggested Readings

The annotated bibliography below gives citations to the primary papers from the KEDS project, as well as some surveys of contemporary event data analysis and computational methods for processing natural language. The Appendix: Event Data in International Relations Research at the end of the manual provides an extensive general survey of event data analysis.

1.12.1 KEDS

Gerner, Deborah J., Philip A. Schrodt, Ronald A. Francisco, and Judith L. Weddle. 1994. "The Machine Coding of Events from Regional and International Sources." *International Studies Quarterly* 38:91-119.

- Description of the DDIR-sponsored KEDS research; includes tests on German-language sources and a foreign affairs chronology.

Schrodt, Philip A. and Deborah J. Gerner. 1994. "Validity Assessment of a Machine-Coded Event Data Set for the Middle East, 1982-1992." *American Journal of Political Science* 38:825-854.

- Statistically compares KEDS data to a human-coded data set covering the same time period and actors.

Schrodt, Philip A. , Shannon G. Davis and Judith L. Weddle. 1994. "Political Science: KEDSA Program for the Machine Coding of Event Data." *Social Science Computer Review* 12,3: 561-588.

- General description of KEDS and an extended discussion of some of the problems encountered coding Reuters.

1.12.2 Event Data

Duffy, Gavin, ed. 1994. *International Interactions* 20,1-2

- Special double-issue on event data analysis.

Merritt, Richard L., Robert G. Muncaster, and Dina A. Zinnes, eds. 1994. *Management of International Events: DDIR Phase II*. Ann Arbor: University of Michigan Press.

- Reports from the DDIR projects.

1.12.3 Computational Methods for Interpreting Text

Advanced Research Projects Agency (ARPA). 1993. *Proceedings of the Fifth Message Understanding Conference (MUC-5)*. Los Altos,CA: Morgan Kaufmann.

- Reports from a large-scale ARPA project on developing computer programs to interpret news reports on terrorism in Latin America; these use a variety of different techniques.

Pinker, Steven. 1994. *The Language Instinct*. New York: W. Morrow and Co.

- Excellent non-technical introduction to contemporary linguistics; extensive discussion of the problems of parsing English

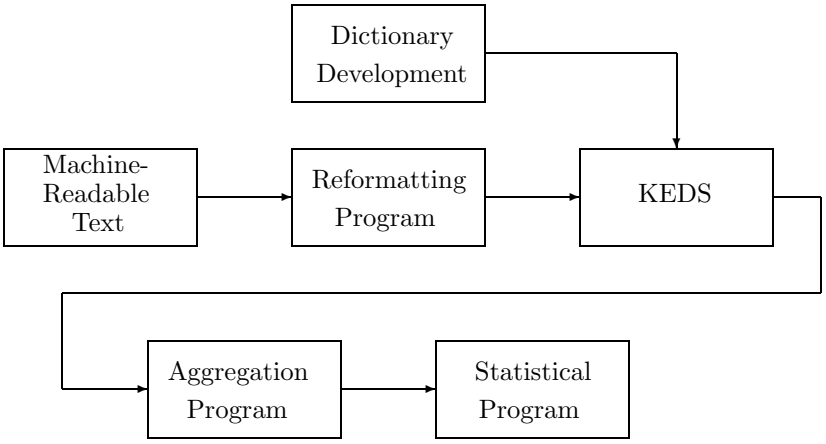
Salton, Gerald. 1989. *Automatic Text Processing*. Reading, Mass: Addison-Wesley.

- General introduction to the use of computers to process text; covers a wide variety of methods.

Chapter 2

GENERATING DATA WITH KEDS

Because both the natural language text *input* and the event data *output* of KEDS are uncommon in the standard statistical data processing environment, working with KEDS requires a few more steps than, say, pulling variables off the Euro-Barometer surveys. The figure below shows what is involved in going from machine-readable text to data that can be analyzed with a statistical program.



2.1 STEP 1: Locate and reformat a set of machine-readable texts

The very first step in doing research with KEDS is finding a source of machine-readable text. This will usually come from an on-line data service or CD-ROM.[1]

In all likelihood, the original text will not be in the input format used by KEDS. For example, the text provided by the Nexis data service that we've been using originally looks like

```
] ] ]
]                            LEVEL 1 - 5 OF 914 STORIES
] ]             Proprietary to the United Press International 1981
] ]                      <December> 29, 1981, Tuesday, PM cycle
] ]HEADLINE: World News Summary
] ] BODY:
]   Sen. Charles Percy, R-Ill., chairman of the Senate Foreign
]Relations Committee, asked his Israeli hosts to avoid the annexed
]
^
^
]Golan Heights in a helicopter inspection tour today of the tense
]Israeli-<Lebanese>frontier.
]
```

This needs to be converted to the KEDS input format

```
811229UPI001
Sen. Charles Percy, R-Ill., chairman of the Senate Foreign
Relations Committee, asked his Israeli hosts to avoid the annexed
Golan Heights in a helicopter inspection tour today of the tense
Israeli-Lebanese frontier.
```

We've developed a number of reformatting programs that remove all of the irrelevant information found in the Nexis download and reformat the text; these programs and their source code (in Pascal and C) can be found at the KEDS web site.

However, unless your original text is in exactly the same format as we found with NEXIS, you will need to write your own filter or modify one of ours. Because

[1] In Gerner et al 1994 we discuss our experiments with using a scanner and optical character recognition (OCR) to generate machine-readable text from printed documents. We had very mixed results with this but OCR software was relatively primitive at that point and the technology has apparently improved substantially in recent years.

machine-readable data are usually consistently formatted, this is usually not very difficult provided you know (or know someone who knows) a programming language such as BASIC, Pascal or C, but this filter is a necessary step before you can start using KEDS. If you aren't able to get a filter/reformatting program written in a program language, the macro language in Microsoft's *Word* program provides another possibility for reformatting.

2.2 STEP 2: Develop the initial coding dictionaries

KEDS uses large dictionaries of proper nouns and verb phrases to code the actors and events it finds in the source text. If you intend to code political events, you would probably find it easier to modify the dictionaries that have been developed by other projects than starting with a new dictionary.[2] The advantage of this approach is that the existing dictionaries have already identified most of the English vocabulary used in Reuters, so even if you expect to substantially change the coding scheme, you will know what types of phrases to expect. The KEDS (WEIS, Middle East) and Pevehouse (BCOW, Middle East) dictionaries are available from our project and are archived at the ICPSR; the PANDA dictionaries are available from the PANDA project (contact: dbond@cfia.harvard.edu). All three of these dictionaries were based on coding Reuters lead sentences. The **Actor_Filter** utility program – available at the web site – can be useful in identifying new actors that are not already incorporated into the dictionaries.

2.3 STEP 3: Fine-tune the dictionaries

With the initial dictionaries incorporated into your system, the next step is fine-tuning "tweaking" the phrases to work correctly with your data and coding scheme. This is done by going through a large number of texts and modifying the vocabulary as needed; this process will also give you an indication of the accuracy of the system. Most vocabulary modifications involve the addition of specific individual actors (e.g. political leaders; geographical place names) and the addition of verb phrases describing behaviors specific to the problem you are considering.

While you are fine-tuning the dictionaries you might also look at some of the advanced features of KEDS, such as the use of substitution rules, word classes, the complexity filter, and additional coding features such as issues and content analysis counts. The grammatical transformation rules may enable you

[2] The KEDS and PANDA dictionaries are each the result of abut two person-years of coding, so a substantial amount of effort has already been invested in developing them.

to develop general solutions to problems that would otherwise require a large number of specific phrases, and the additional coding features allow information to be extracted from a sentence beyond the basic *source-event-target* structure of event data.

> **Note:** Resist the temptation to tweak the vocabulary indefinitely! Tweaking should focus on finding *general* patterns that will occur on multiple occasions in the text, not on expanding the list of phrases to cover every possible contingency. Always remember that coding errors will be only *one source* of noise in your data: the source text is already an incomplete and biased record of the underlying events; the event coding scheme may be incorrectly aggregating some categories of events; and the statistical models in which the data will eventually be used in capture only some of the possible forms of the political relationships.
>
> Even if your coding was somehow "perfect," you are still dealing with a noisy process. Successfully identifying relationships amid that noise comes at the *analytical* stage of the project – both the development of the coding scheme and the statistical analysis – not in the coding stages. Know when the coding accuracy is "good enough," and don't fall into the trap of producing a project that does beer-budget analysis on champagne-budget data. If you can't cope with the fact that probably 15% of your data are erroneously coded,[3] you shouldn't be doing event data analysis. End of sermon.

2.4 STEP 4: Autocode the entire data set

Unless you intend to use KEDS for machine assisted-coding of your entire dataset, the data should be autocoded once the accuracy of the dictionaries has reached a level you are comfortable with. Autocoding will ensure that the coding rules have been consistently applied across the entire data set, rather than having the part of the data that was used to develop the dictionaries coded by hand, and the remainder machine coded. Autocoding also insures that your coding can be replicated by later researchers, as well as by yourself at a later date.

If you cannot get the accuracy level of KEDS to an acceptable level, you still may be able to use the program for routine coding by using KEDS's complexity filter. This will automatically – and systematically – divert to a separate file any texts that appear too complex to machine code (for example texts containing an excessive number of verbs or actors, or containing ambiguous words such as

[3] And this is true whether you are using human or machine coding.

ATTACK or GEORGIA). The complex texts can be processed using machine-assisted coding, and then the two sets of event data can be merged. Coding with the complexity filter is not completely replicable – and the complexity filter will not catch sentences that might be coded incorrectly – but it is more efficient and replicable than all-human coding, but more accurate than all-machine coding.

2.5 STEP 5: Aggregate the data for statistical analysis

KEDS produces standard event data of the form

<DATE><SOURCE CODE><EVENT CODE>
<TARGET CODE><AUXILIARY CODES>

Because event data are an irregular, nominal-measure (categorical) time series they must be aggregated before they can be used by standard statistical programs such as SPSS and SAS or graphical displays such as spreadsheets; these all expect a regular, interval-measure (numerical) time series. This transformation is usually done by mapping each event code to an interval-level scale (for example, Goldstein 1993), and then aggregating the data by actor-pair and week, month or year using averages or totals.

It is *possible* to do this aggregation by scripting the data transformation facilities of a statistical program. However, this process tends to be very slow and awkward, particularly when dealing with a large number of actor pairs. As an alternative, we have developed an aggregation program, KEDS_Count, to automate this process; this program and its documentation are on the KEDS disks. In contrast to the text reformatting programs, which need to be customized, KEDS_Count should handle most situations requiring aggregation of event data into a time-series.

2.6 Using this manual

The remainder of this manual is divided into two parts, plus some appendices. Chapters 3 through 12 of the manual deal with the development of coding dictionaries and various optional features affecting the operation of the program. Chapters 13 and 14 describe the actual operation of the program, including the windows, menus and program controls. The appendices cover error messages, the numerical limitations of program and an extended discussion of event data.

If you are just trying to get started with the program, read through the sections in Chapters 3 through 12 on Input Files and .Verbs and .Actors Dictionaries, skim

Chapters 13 and 14 and work through the KEDS.Sample demonstration. You should also look through the long *.Verbs* dictionary to get an idea of how KEDS deals with verbs and verb phrases. Many of the optional features of KEDS – for example Output Formatting, Issues and Agents – are needed only in specialized applications and the program works fine without them.

Chapter 3

INPUT FILES

3.1 Purpose

This chapter describes the files used by KEDS. These are:

.Project: The project file contains information about all of the other files used in the coding, as well as a coding history.

.Text: The source text to be coded.

.Verbs: A file containing a dictionary of the verbs and verb phrases used to code events.

.Actors: A file containing a dictionary of the nouns used to identify the sources and targets of events.

.Options: A file containing a variety of commands that modify the behavior of the KEDS program.

.events: An output file containing the coded event data; this can be formatted in a variety of ways.

All of the files except for the *Project* file are in "flat ASCII" format (Macintosh TEXT-type) and should only be edited using a program that produces a flat ASCII file (e.g. a file without embedded control codes). The *Project* file is maintained by the KEDS program itself and cannot be edited.

In addition to the required files, KEDS also can work with the following additional files; the details of the files are described later chapters.

Input

.Issues: These files contain character strings used to identify the context of the text [see chapter on **Issues**]

.Classes and Rules: This file defines classes of words and transformation rules used on the text [see chapter on **Classes and Rules**]

Output

.probs: record of problems encountered in the data. [see chapter on .Options]

.complex: source text that was too complex to machine-code by machine [see chapter on .Options]

.text: tab-delimited file containing the source text, date and sentence identification [see chapter on Formatting Output]

3.2 Project File

KEDS uses a "project file" containing information about the names of the *.Verbs, .Actors, .Options* and text files, and the status of the Valid and Pause When settings. It also tracks up to 64 coding sessions, recording the coder, time and date, number of phrases examined and the accuracy of the system during that session. The contents of the project file can be displayed or written to a text file using the **DISPLAY/Status/Project** option in the menu.

3.2.1 Initializing a Project

To create a new project file or reset an existing file, click the **Initialize new project file** option at the beginning of the program. This option presents a series of Macintosh file selection dialogs that are used to select – in order – the text, *.Verbs, .Actors* and *.Options* files for the project. The type of the requested file is shown in the upper left corner of the dialog.[1] The files associated with a project file should all be in the same folder. Clicking Cancel during any of these dialogs will stop the initialization and quit the program without making any changes.

After the introductory sequence of file selection dialogs, a file name can be changed by clicking the label to the left of the box containing the file name: this will give a standard Macintosh file selection dialog. The name can also be changed directly by editing, though this is less reliable that using file selection.

[1] Usually: some Macintosh extensions (INITs) move or enlarge the standard file selection dialog box, and this may obscure the message indicating that type of file is being requested.

The Project Name box provides a prefix that is used for the *.events* and output files: enter the name in the box provided. Keep the name short (e.g. less than 16 characters) because file suffixes and serial numbers will be appended to it to create other file names. This prefix also becomes the project name. If a project already exists with this name, you will get an alert dialog asking for verification that you want to reset the file.

Note: The text file is actually optional – if you enter a null string as the text file, the project file will cause the program to prompt you to select a text file each time it is started. To enter a null file name, first select any text file during the initialization (hitting Cancel will just quit the program), then click in the text file name edit box, then hit <Return>. Null text file names are particularly useful when you want to autocode a number of texts using the same set of dictionaries.

3.3 Text File

KEDS will process with any TEXT file containing records in the following format

date idinfo any other information
source text line 1
source text line 2 ...
source text line n
blank line

Your filter program should reformat your original text into this form, or you can edit the original text manually. If you are using a word processor, be sure to save the file in the TEXT format rather than the normal format of the editor.

date is the date of the sentence being coded and should be 6 digit in the form YYMMDD and should be in the first six characters; for example 880318=March 18, 1988. See comments below under Date Restricted Codes for the interpretation of dates that are out of range and dates in the years 2000 to 2010. If the **date** string is less than 6 characters, it will be set to 140101.

idinfo is the next six (6) characters following the date and can anything used to identify the source text (including blanks); typically it is a serial number that uniquely identifies the source text in a specific day. This can also be used to identify whether records refer to sequential sentences; see the discussion of the FORWARD command in the *.OPTIONS* chapter.

text lines should be around 80 characters in length; after filtering punctuation, any characters in excess of 96 are ignored.[2] The source text is limited to

[2]In other words, the maximum length of a line is 96 characters. However, if a 96-character

16 lines containing a maximum of 255 words (if the number of words is greater than 255, the remainder will be ignored) and a maximum of 2560 characters.

blank line is a line containing no characters (in other words, just a <return>), or containing only blanks.

3.3.1 Utility Programs

In addition to KEDS, our project uses several utility programs for the pre-processing of source texts and the post-processing of event data. All of these programs are available at the web site.

NEXIS_Filter

Over the years we have developed a series of different filter programs that convert NEXIS downloads into the appropriate format for processing by KEDS. The most recent version is written in C and will manage both lead-sentence filtering and full-story filtering, though only for Reuters stories. The source code as well as the compiled program is posted at the web site. Earlier versions (with source code) are available in Pascal, and some of these handle wire sources other than Reuters.

Managing the source text for a long time series is a major task, and if you are not familiar with the Macintosh, you might want to do these on a Window or Unix platform. Because KEDS input and output are both in flat ASCII files, this is not difficult, and several projects (notably PANDA) have worked with KEDS in cross-platform environments.

ACTOR_FILTER

This program locates potential new actor names in a file of KEDS input records by looking for strings of consecutive capitalized words and comparing these against an existing sets of actor names and a list of stop words. The output of the program is a keyword-in-context (KWIC) file sorted by the frequency of the actor name. The program basically runs in batch model, controlled by an optional text file.

We've used this program to develop a number of new actor dictionaries for coding internal events, and we've found that it significantly decreases the time

line contained commas that were not surrounded by spaces, filtering would cause it to become longer than 96 characters and the end of the line would be lost. To be on the safe side, keep the individual lines around 80 characters.

required to modify the *.Actor* files. The KWIC index typically has quite a few false positives – in particular names that are preceded by distinct descriptive adjectives – but it is a fast way to figure out who is who in a new set of data. The program is in beta status, but seems to be stable. The compiled program is posted in the "Software" section of the web site.

NEXIS_VERIFY

This utility program goes through a list of KEDS records and checks the dates for missing intervals, bad date formats and the like. Date strings are tagged on any of the following conditions:

1. Date is not in the range 790415 to 970610, the current NEXIS limits on Reuters
2. Consecutive dates are separated by more than 4 days
3. A date occurs before the date of the previous record.

These conditions can be modified by setting parameters in the source code. Both the compiled program and the C source code are posted.

Based on long experience with the vagaries of Reuters and NEXIS records, we strongly recommend running this routine before coding, particularly if you subsequently intend to aggregate data using KEDS_Count.

KEDS_Count

This program aggregates event data by time period and dyad. The program was originally designed to work with output from KEDS but will work with any tab-delimited event data. The program operates in a "batch" mode by first reading a "command file" – created using a word processor – that defines the characteristics of the aggregation. It then reads through the event data files and produces a tab-limited output file for each dyad. These output files can be read into a spreadsheet or statistics program.

3.3.2 Automatic Input Filtering

Unless the **SET: FILTER INPUT=FALSE** option has been set in the *.Options* file, the following additional changes are made in the text before it is processed:

1. All of the search-and-replace patterns specified by the REPLACE options are applied.

2. All characters are converted to upper-case and any diacriticals are removed (i.e. ö becomes O, å becomes A, é becomes E and so forth).[3] This step is **always** done.

3. Periods following lower-case letters are replaced with blanks; this means that the punctuation at the end of a sentence is eliminated but periods in abbreviations (e.g. U.N.) are retained.

4. '?' and '!' are replaced with blanks.

5. Commas inside numbers are eliminated, so "1,500" becomes "1500".

6. Semicolons (';') are replaced with commas

7. Commas and double-quotes (") are delimited with blanks, so the sentence

   `Clinton◇responded,◇‘‘I don’t know.’’`

   becomes

   `Clinton◇responded◇,◇‘‘◇I don’t know◇’’◇`

   where ◇ represents a blank

All of this filtering is done before the text is entered into the system, so these changes will be reflected in the text displayed on the screen. Generally KEDS formats the source text to fit the size of the text window displayed on the screen. If you want to have a "hard-coded" return in the text, use "◇^ /◇" (note the spaces before and after "^ /")

If the **SET: FILTER INPUT=FALSE** option is used, then only [2] is done: all characters are converted to upper-case and any diacriticals are removed.

Any text between the delimiters /*...*/ will not be coded; this allows irrelevant material (for example subordinate clauses) to be eliminated from coding and for comments to be added to the text. Additional sets of delimiters can be specified using the **OMIT** command in the *.Options* file.

Notes: KEDS does nothing with the following punctuation issues:

1. Hyphenated words are the end of a line, for example

```
                ..... confronting antidisestab-
        lishmentarianism, the government .......
```

[3] Prior to Version 0.7, KEDS accommodated diacriticals. These can still be handled by using the **REPLACE** option to replace them with a letter combination that does not use diacriticals (e.g. Ä $->$ A) and then using those combinations in patterns.

These should be combined into single words when filtering the original text.

2. Any other forms of hyphenation, e.g. “Arab-Israel”, “al-Asad”

3. Apostrophes, e.g. “didn’t”, “Shi’a”

4. The asymmetrical “smart quotes” “ and ” if these occur in your source text, replace them with a simple double-quote " using a word processor or a REPLACE command in the .Options file. The same goes for “smart apostrophes”.

3.3.3 Coding a number of files

Usually the Text File box contains the name of a single file. There are two exceptions to this:

If the entry in the Text File box is ***empty***, then every time the project file is run, the program will present a Macintosh file selection dialog asking for the name of the file to be coded. This allows a number of files to be coded using the same set of dictionaries.

If the first line of the file in the Text File box contains the word

AUTOCODE

then the file is assumed to contain a list of file names. For example:

AUTOCODE
LEVANT.89.LEADS
LEVANT.90.LEADS
LEVANT.91.LEADS
LEVANT.92.LEADS
LEVANT.93.LEADS

Each of these files will be autocoded without any pausing or dialogs in the sequence listed. This allows a large set of files to be coded without supervision in a “batch” mode. If a file cannot be found, the program presents an alert dialog to that effect, then terminates.

Chapter 4

.VERBS AND .ACTORS DICTIONARIES

4.1 Purpose

Most of the coding decisions made by KEDS are based on the dictionaries of words and phrases in the *.Verbs* and *.Actors* files. This chapter describes the basic features of those dictionaries.

While the dictionary files can be modified using dialog boxes while the program is running, KEDS is not designed to build those files from nothing: working with empty *.Verbs* and *.Actors* files will have unpredictable results. It is generally more efficient to initially create or extensively modify the dictionaries with a word processor rather than building the entire vocabulary using the **Modify** menu option inside the program.

In order to work with the dictionaries (as well as the *.Options* file), you will need to be able edit files using a word processor or editor that produces flat ASCII, Macintosh **TEXT** - type files. Virtually all word processors can do this, but be sure that at every stage of your processing, the files that KEDS will be reading have been saved as **TEXT** rather than in a word processing format. If a file that was working earlier suddenly starts producing garbage or crashing the program, it was probably accidentally saved as a formatted document – for example *Microsoft Word* will do this if you inadvertently format any part of the document.

4.2 Phrase Input Formats for .Verbs and .Actors files

The .Verbs and .Actors files use the same format:

PHRASE [***CODE***] **{;** ***comments*****}**

The last line in the file should be ~ **FINISH**; this is the internal end-of-file mark. Lines after the ~ **FINISH** will not be read into the dictionary but will be copied into the saved file.

Initial and terminal blanks are trimmed from ***PHRASE***. The comments after the semicolon are optional; this allows the origin of the pattern (e.g. the coder identification and date) to be documented.

Any line *beginning* with \ ("backslash") is a comment and ignored. This can be used to temporarily eliminate phrases from the dictionaries.

4.2.1 Codes

Codes are less than or equal to 12 characters in length and can contain only letters and numbers. (i.e. no special characters such as '!', '[', or '$'; in particular avoid the special codes - - -, ***, ### and +++). Although 12 is the maximum length of a code, most of the default display sizes and the editing boxes in dialogs are formatted for 6 character codes. The length of a code should to balance the mnemonic advantages of long codes with the difficulties of using lengthy codes in complex code constructions and the increased size of the resulting output files.

4.2.2 Stemming

Stemming can be used to match different forms of the same word using a single string called a "stem". For example

ACCEPT:	**ACCEPTS**	**ACCEPTED**	**ACCEPTING**
SYRIA:	**SYRIA'S**	**SYRIAN**	**SYRIANS**

KEDS handles stemming by matching from the beginning of the word. A word is considered to match a stem provided every character in the stem matches. In other words, **SYRI** will match all four forms of **SYRIA** but it will not match **SYRACUSE**.

When phrases have the same initial letters, KEDS checks long phrases before shorter ones: For example **SIGNALLED, SIGNED and SIGN** have the search order:

SIGNALLED
SIGNED
SIGN

To prevent a character string from being used as a stem, put an underscore (_) after the string: this means that the string will match only if it is followed by a space. The phrase **OF_** will only match "`OF◇`" whereas the stem **OF** would match "`OF`", "`OFFER`" and "`OFFICIAL`."

Considerable effort may be involved in the dictionary tweaking to determine which stems can be used without causing problems. For example **TOUR** looks like a useful stem for TOUR, TOURED and TOURING, but it also matches TOURIST, which can cause problems. The phrase:

HEAD

- * FOR

correctly handles the verb forms `HEADING FOR` and `HEADED FOR` but inconveniently also matches `HEADQUARTERS FOR`. To force an entire word to be used in a match, end it with an underscore; alternatively, problematic words such as `HEADQUARTERS` can be eliminated using null codes (see below).

> **Note:** In our experience, stemming is the most frequent cause of wildly inaccurate coding errors, and it is not entirely clear that this feature is worth the trouble. In a recent case, for example, the verb **BEAT** – which is generally problem-free – matched BEATY, the name of an American released by Iraq. A future version of KEDS may incorporate a facility for designating regular verb constructions and noun endings (e.g. plurals and adjectival forms) and make stemming optional.

4.3 .Actor Dictionary

The *.Actors* file contains proper nouns their associated codes. Multiple nouns can refer to the same code; for example in our system ISR (Israel) corresponds to `ISRAEL, ISRAELI, ARENS, PERES, JERUSALEM, TEL AVIV` and `SHAMIR`.

Example
ABU_SHARIF [PLO] ; tony 3/13/91
ACQUINO [PHL]
AL-WAZIR [PAL] ; tony 3/13/91
AMMAN [JOR]
AMNESTY_INTERNATIONAL [NGO] ; tony 3/13/91
ANKARA [TUR] ; tony 3/13/91

ANTIGUA [ATI]
AQUINO [PHL] ; tony 3/13/91
ARAB [ARB]
ARAFAT [PAL]
ARCHBISHOP_OF_CANTERBURY [PVO] ; tony 4/21/91
ARENS [ISR]

It is occasionally necessary to use two or more spellings for a single actor, particularly if multiple (or poorly edited) text sources are used, as with `ACQUINO` and `AQUINO` in the example above (to say nothing of `QADDIFY, GADDIFY, QADHFI, KHADAFI`...).

See Also: Coded Compound Actors, Date Restricted Codes

4.4 .Verbs Dictionary File

The .Verbs file contains a combination of simple verbs (e.g. PROMISED) and verbs plus associated words (e.g. PROMISED FUNDS). For example

ACCEPT
- PROPOSAL WAS * [081]
- * FORMULATION [041]
- REFUSED TO * [112]
- * INVITATION [082]
- * PROPOSAL [081]
- * CHARGES [013]

The root verb is ACCEPT and it will match ACCEPT, ACCEPTS, and ACCEPTED. The phrases that start with "-" are the associated phrases along with their codes, e.g.

```
ACCEPTED PROPOSAL
```

will be coded 081 (WEIS "agree") while

```
REFUSED TO ACCEPT
```

will be coded 112 (WEIS "reject"). The phrases tend to involve the direct object of the verb, though this is not always the case. The "*" indicates where the verb itself should appear, so the first phrase would match

```
PROPOSAL WAS ACCEPTED
```

If a verb by itself uniquely identifies a code – which is commonly the case for WEIS – it is on a line by itself along with the code. A verb can also have a default code followed by exception phrases:

AGREE [081]
- * TO_LET_ESCAPE [066]
- * LOAN [071]
- * AID [071]

In this case, the verb **AGREE** normally codes a 081 ("Agree") category but the `AGREED TO LOAN` would code 071 ("Extend economic aid") and `AGREED TO LET ESCAPE` would code 066 ("Release persons or property").

By default, KEDS tries to match phrases in the order of longest to shortest; phrases are automatically sorted by length as they are entered. If you want the phrases checked in a different order, this can be done by either

- Change the order in the .Verbs file using a word processor;
- Insert some additional blanks in the middle of the pattern to make it longer than the phrases you want checked after it. These blanks will be removed as the phrase is recorded, but the phrase will be positioned in the list according to its length with the blanks included.

Note: The verb token * shows the location of the verb root as a separate string and it cannot be attached to prefixes, suffixes or verb endings. The only characters that can legally precede and follow the * are the underscore and blank. Other characters will cause a syntax error. Prefixed and suffixed forms of verb roots (e.g. PATROL/PATROLMAN) should be entered as separate, null-coded roots.

4.4.1 What is a verb?

In a formal language such as Pascal or SAS, words are associated with a single meaning or a small set of related meanings. While this is often true when dealing with natural language – for example the English words *accuse* and *deny* are almost never incorrectly coded in Reuters leadsthere are exceptions. In English, there are ***lots*** of exceptions.

Those exceptions are due to the fact that English is primarily an "isolating" language where the grammatical role of a word can change depending on its position in a sentence.[1] For example, in the phrase

[1] See Pinker (1994 , chapter 12) for an extended discussion of these issues. The problem is further complicated by the fact that English is derived from an inflected Germanic language that evolved over a millenium into a language that is now largely isolating. Vestiges of inflection remain at quirky points in the language – for example "correct" English retains (barely...) the distinction between the nominative *who* and the accusative *whom* while dropping that distinction in *you/ye.*, and the inflected "-ed" is used to indicate past tense.

```
The treaty was broken
```

the word *broken* is a verb, while in the sentence

```
The diplomats discussed the broken treaty
```

, the word *broken* is an adjective. In contrast, in inflected languages – for example Latin, Russian or American Sign Language – a root is usually modified by prefixes, suffixes or vowel changes to indicate that it is being used for a different purpose. For example, in Latin either of these phrases correspond to "Man bites dog":

```
Homo canem mordet
Canem homo mordet
```

The order of the subject and object are irrelevant; to create the sentence "Dog bites man," the nouns themselves must be changed:

```
Canis hominem mordet
Hominem canis mordet
```

English, in contrast, uses the same word whether a noun is a subject or object[2] and the role of the noun as subject versus object is determined by its position. Those positions can be changed by the use of passive voice:

```
The dog was bitten by the man
```

but "dog" and "man" are still unchanged.

Words can also change from verbs to nouns without modification:

```
When Jill returned from the car wash, she parked her car in the drive.
```

The only indication that *wash* and *drive* are nouns rather than verbs comes from their position in the two prepositional phrases.

As Pinker notes, English is a language containing a extraordinary number of homonyms – words that sound identical but have different roles and meanings

To further complicate matters, a series of arbitrary grammatical rules derived from Latin (an inflected language) were incorporated into formal English during the 18th century by socially-mobile London elites seeking to differentiate their use of the vernacular from that of the masses: The two notorious examples of this is the prohibition against split infinitives and against ending a sentence with a prepositions. These two constructions are completely consistent with the underlying grammar of English, but not Latin.

In the late 20th century, these rules were incorporated into the so-called "grammar checkers" found in word processors. Those programs contain parsers that are somewhat more sophisticated than the one used in KEDS but they are still incapable of handling many of the common grammatical techniques mastered by 5-year-olds. As a consequence, the grammar checkers look for the easy stuff – split infinitives, passive voice – and cybernetically complain about it. The resulting text tends to have the same relationship to natural language that elevator music has to Mozart.

[2]It retains this distinction in pronouns: "I" versus "me", "she" versus "her" and so forth.

depending on how they are used in a sentence. This fact has yet to filter down into popular discussions of the language – hence the ad nauseam complaints about the alleged misuse of *hopefully* at the beginning of a sentence

```
Hopefully, the ceasefire will hold
```

when in fact the role of *hopefully* is **not** to modify a verb in this sentence[3], any more than the role of *wash* is that of a verb in the phrase *car wash*.

What does this mean for KEDS? It is not a full parser, and generally assigns each string of characters to only a single class of words.[4] This can lead to ambiguities. However, KEDS's patterns are oriented towards looking at the location of various words with respect to each other, and this facility is very important in determining the meaning of words in English. The upshot is that you will find that dictionary development involves a lot of effort in finding phrases that must be null-coded[5] or otherwise assigned an interpretation distinct from the root. This is not a feature of the program but rather a feature of the English language. On the positive side, after you have worked with KEDS for some time, you will have a much deeper appreciation of how the grammar of English actually works, as distinct from the much more simplified grammar you were probably taught at some point.

The key to the effective use of phrases is to make sure that phrases are sorted into the transitive verb that determines the event code. Do not code indicators of tense (e.g. HAS, WILL) or forms of "to be" (IS, WAS etc.) as verbs; use the transitive verb that indicates the action (this will often be an infinitive, e.g. in `WILLING TO NEGOTIATE` the verb is `NEGOTIATE` rather than `WILLING`). Having done that, the next step is to null-code phrases where the word is not being used as a verb, either because it occurs in an idiomatic expression ("Read my lips") or because the string matches a homonym that is not a verb.

In our work with Reuters reports on the Middle East, two words stand out as particularly problematic: FORCE and ATTACK. Both words can be used either as nouns ("A guerrilla force launched an attack") or as verbs ("Rebel radio said guerrillas would attack in order to force concessions") and occur frequently in reports about military conflict. FORCE and ATTACK are further complicated because they can be used to refer both to verbal actions (persuasion and criticism) and to uses of force; both uses are common in Reuters. In our dictionaries, a large number of patterns are associated with each of these words to try to distinguish the noun usage from the verb usage. ARMS, BATTLE, FIRE, HELP, ORDER, PLAN, PLEDGE, STRIKE and SUPPORT are other

[3] It expresses the intent of the speaker, and as such is comparable to "accordingly," "generally," "ideally," curiously," "supposedly" or any of the 18 additional examples in Pinker's list of "-ly" words that modify sentences.

[4] This can be overridden using a CLASSES: statement, discussed in the Classes, Composite Patterns and Rules chapter.

[5] English also, of course, allows verbs to be created from nouns:
null-cod'ed. *verb.* 1. the act of assigning a null code [- - -] to a verb phrase in KEDS.

examples of words that are used both as verbs and as nouns.

A final complication arises from the tendency of the English to based some adjective on verbs – for example "the wrecked car" or "the broken treaty." Usually these do not cause problems – in fact at times they allow KEDS to assign the correct code based on an adjective rather than a verb – but they should be kept in mind.

4.4.2 Pattern Matching in Verb Phrases

Verb phrases can use either simple or composite patterns. This section will discuss the simple patterns; composite patterns are covered in the chapter Classes, Composite Patterns and Rules.

By default, phrase matching stops when a conjunction (AND_, BUT_ and any additional conjunctions specified in the *.Options* or *.Class* file) is encountered, so that a verb phrase cannot match words that occur after a conjunctions. The word AND_ in a compound actor phrase does not affect this, since it is removed when the compound actor is constructed. There are two exceptions to this rule:

- The conjunction is connected to verb by underscores, e.g.

  SHOT
  - *_AND_KILLED[6]

- The parameter **IGNORE CONJUNCTIONS=TRUE** has been set in the *.Options* file.

When coding expository written text such as news leads, using KEDS default treatment of conjunctions will usually produce the best results. **IGNORE CONJUNCTIONS=TRUE** is sometimes helpful when coding transcribed verbal material (e.g. news conferences), where conjunctions are used more frequently.

[6]In general, this type of construction should be avoided because the conjunction AND will not be eliminated. If a conjunction is used in a compound verb, include both parts as a distinct verb , rather than as a phrase, in order to keep the sentence from being coded as compound. Compound verbs become particularly problematic when a phrase occurs before the verb the construction

KILLED [678]
- SHOT_AND_* [123]

SHOT [345]
creates two separate events:
SHOT [345]
SHOT_AND_KILLED [123]
because the SHOT_AND_* patterns is matched across the conjunction, and SHOT is coded because it is in the first part of a compound sentence.

Notes: Conjunctions do not stop searches for generic actors , but conjunctions do stop the search for a *designated* actor using the $ and + options in a phrase. (see below). The *.Options* file command

SET:IGNORE CONJUNCTIONS=TRUE

deactivates the conjunction-limited phrase matching. See comments under Compound Sentences.

Problems matching words inside of multi-word phrases:

Due to a quirk in KEDS pattern matching algorithm, if a word is part of a multi-word phrase connected by underscores:

STEALTH_FIGHTER [USA]

it can will no longer match in a verb phrase such as

FLEW [031]
- FIGHTER * [223]

With this combination, the sentence

```
Stealth fighters flew towards Baghdad...
```

would code to

USA 031 IRQ

rather than

USA 223 IRQ.

This characteristic of the algorithm is closer to a bug than a feature, but unfortunately it is deep in the code and would be difficult to correct. The verb phrase

FLEW [031]
- STEALTH_FIGHTER * [223]

will get around the problem, though it isn't a general solution. A future version of KEDS will probably correct this.

Skipping Intermediate Words

By default, pattern matching skips over intermediate words: the phrase

PROMISED
- * DOLLARS

will match the phrases

```
PROMISED TWO MILLION DOLLARS
PROMISED A GRANT OF 30-MILLION DOLLARS
```

If two words *must* be consecutive, connect them with an underscore, e.g.

BURNED_DOWN

The underscore character and concatenation can be used to force a word or set of consecutive words to be directly before or after the verb; this is particularly useful for words that can be interpreted as either nouns or verbs. For example:

ATTACK [**122**]
- * CRITICISM OF [**042**]
- *_HELICOPTERS [- - -][7]
...
POUND [**223**]
- BRITISH_* [- - -]

Designated Actors

Phrases can specify the location of the source and target by using the tokens **$** and + respectively. For example, the phrase

ADVISE
- +_WAS_*_BY_$

would do the correct assignments on the phrases

```
EGYPT WAS ADVISED BY THE UNITED STATES
```

$ and + are assigned to the first actor that is encountered in the appropriate location. In other words in matching

- + WAS * BY $

the system will search for the target using the first actor before WAS ... ADVISE and search for the source using the first actor after BY. In this example, the actor locations are used to reverse the source and target when a verb phrase is in passive voice.[8]

The symbol **%** specifies a *compound actor* that should be assigned to both the source and target; it works with either coded or parsed compounds. This is typically used when dealing with consultations:

[7] "- - -" is a "null code" that causes a phrase *not* to be coded; it is described below.
[8] This could also be done using a general rule for passive voice; this is described below.

```
REPRESENTATIVES OF SYRIA AND JORDAN WILL MEET IN CAIRO
```

The phrase

MEET
- % * IN [031]

would do the correct assignments

SYR 031 JOR
JOR 031 SYR

rather than

SYR 031 EGY
JOR 031 EGY

If no compound actor is found in the sentence, the phrase containing the compound assignment fails, and then any shorter phrases in the verb's phrase list will be checked.

> **Note:** If a sentence might have both a compound subject and a compound object:
>
> `Representatives of the USA and Israel will meet with representatives of Syria and Lebanon`
>
> be sure that a '$ * +' comes before the '% *' pattern
>
> **MEET**
> **-$ * +**
> **-% ***
>
> If this is not done, the % token will cause the actors in the compound subject to be used as both the source and target, ignoring the compound object.

The source and target tokens are optional; if they are not specified then default rules (see below) are used to locate the source and target. However, if a source or target is specified in a phrase and then not found in the appropriate location, the phrase as a whole fails. Similarly, if a string in a phrase is missing, then the phrase as a whole fails.

> **Note: Precedence in phrase matching**
>
> A phrase is matched in the forward direction from the beginning of the clause.[9] Each element of the phrase is matched in order. For example, if one had a sentence of the form

[9]This is a change from the system used in KEDS prior to version 0.9, where pattern matching first looked backwards, then forwards, from the verb. This change does not seem to make much difference in coding.

```
<actor1> s1 <actor2><verb>
```

and the phrase

- + s1 *

the target would be assigned to <actor1>, because this occurs before **s1**.

If the sentence had the form:

s1 <actor2><verb>

the phrase match would fail, because there is no actors prior to **s1**.

Because phrase matches are stopped by conjunctions, a phrase match will only apply inside a clause in a compound sentence.

4.5 Default Actors

If a verb phrase does not designate a source and target, these are filled in using default searches. The default source search is done first: it starts at the beginning of the text and the first actor encountered *prior to the verb* is designated the source.[10] If there is no actor prior to the verb, the source is set to missing. The default target search then tries to find an actor *after the verb* that has a code distinct from the code of the source; if no such actor is found, a search is made for the first actor *before the verb* that has a code that is distinct from the code of the source. If no such target is found and the source is compound, this is also designated as the target; the system automatically eliminates codes which would have the same source and target. Unless the *.Options* file command **SET: IGNORE CONJUNCTIONS=TRUE** has been used, these searches are confined to the conjunction-delimited clause containing the verb. Actors with null codes are ignored in both searches.

If this search fails to assign a source or target and default patterns have been set in the *.Options* file using the **SOURCE** or **TARGET** commands, then the system tries to match one of these patterns. These patterns can be matched *anywhere* in the sentence, not just in the clause containing the verb.

Finally, if no **SOURCE** or **TARGET** patterns match and a default source or target code has been specified using a **SOURCE:** [***code***] or **TARGET:** [***code***] command in the *.Options* file, that code is assigned. If none of these are used, then the item is considered missing and the missing value code *** is assigned.

[10] If a target has already been found through a pattern match, the source is the first actor with a code distinct from the code of the target.

Note: Assigning a default source will guarantee that the first verb in each clause (or the sentence) is coded. Because English words that look like verbs may actually be nouns or adjectives (see the Verbs/What is a verb? section above) this can result in coding that is significantly different from that generated by the default options, where a verb must be preceded by an actor.

Chapter 5

SPECIAL PURPOSE CODES

5.1 Purpose

Most of the codes used in the KEDS dictionaries are the simple codes assigned by the event coding scheme to actors and events. However, KEDS has three special codes – the null, discard and complex codes – that can be attached to phrases to change how KEDS deals with a sentence. Codes can also be assigned priorities, they can be restricted in time, and multiple events can be coded from a single phrase.

The null, discard and complex codes can be can be included in either the *.Actors* or *.Verbs* files. In the *.Actors* file, they can also be included as date-restricted codes. In the *.Verbs* file, they can be assigned to either roots or phrases. These codes do not generate events.

> **Note:** It is generally better to assign discard and complex codes to roots rather than phrases in the *.Verbs* file. When the codes are assigned to a root, a sentence meeting the complex or discard criteria can be identified as soon as all of the words in the text are classified. If the codes are assigned to phrases, the verb roots in the text must be evaluated first, and in some instances the phrases of a root may not be evaluated, for example if the verb occurs late in a phrase or evaluation is halted by a dominant code. The only advantage to assigning the codes in phrases is to keep all of the phrases dealing with a verb in one place. This problem does not apply to null codes, which simply eliminate a phrases from being coded and are very common

5.2 Null Code [- - -]

The null code [- - -] is used to eliminate phrases that would otherwise be confused with actors or verbs. Phrases with null codes can be in either the *.Actors* or *.Verbs* file. Experience in both the KEDS and PANDA projects has shown that finding the appropriate phrases to be null coded is a key element in bringing the accuracy of a coding system above the 75% level. Null-coded phrases are a substantial part of the dictionaries of both projects.

Words that have only a null code – all actors, and nulled-coded verbs without patterns – are converted to type "null" and are henceforth not considered actors or verbs. The choice of putting null-coded words in the *.Actors* or *.Verbs* file is one of convenience and does not affect the efficiency of the coding; usually it is best to put a null-coded word in the list containing words with a similar stem.

Example

Using the actors

ISRAEL [**ISR**]
WEST BANK [**PAL**]

the phrase

`ISRAELI-OCCUPIED WEST BANK AND GAZA`

will generate both ISR and PAL as actors. By adding the null code

ISRAELI-OCCUPIED [**- - -**]

only PAL is generated as an actor.

Example

The phrase

THE HEAD OF LEBANON'S CATHOLIC COMMUNITY

generates a verb identification for HEAD, since is more commonly a verb, e.g.

`EGYPTIAN PRESIDENT MUBARAK HEADED FOR A MEETING WITH`

The code

THE_HEAD_OF [**- - -**]

eliminates this problem.

5.3 Discard Code [# # #]

The text is likely to contain some events which involve multiple international actors but which are non-political: sports events are the most common; traffic accidents and natural disasters involving tourists a close second. These can be automatically discarded by using the discard code [# # #].[1] The text is rejected if the number of discard phrases found is greater than or equal to the **Discard Phrases** parameter in the **Options/Complexity** menu option (default=1) or set using the **COMPLEX:** command in the *.Options* file.

Example
MARIJUANA [# # #]
MICHAEL_JORDAN [# # #]
FRANK_JORDAN [# # #]
REAGAN [# # # >**890120**] [**USA**]
SOCCER [# # #]
WORLD_CUP [# # #]

KILLED [**223**]
- * IN ACCIDENT [# # #]
- WILDLIFE * [# # #]

The presence of a discard condition causes most of the other processing of the text to be terminated; the record will be skipped unless the PAUSE ALWAYS command is active. Discards should therefore be used only when you are certain that the record contains nothing of interest; if you might want to look at it, use a **complex** code (below) instead.

5.4 Complex Code [+++]

The complex code can be used to identify words and phrases in the *.Actors* or *.Verbs* dictionaries that will cause the program to automatically divert the text to a *.complex* file, provided the **COMPLEX:** command has been included in the *.Options* file. The text is diverted if the number of complex phrases found is greater than or equal to the **Complex Phrases** parameter in the **Options/Complexity** menu option (default=1).

For example, the word GEORGIA can refer to a violence-prone region afflicted by ethnic conflict and political demagogues in the southern region of the former

[1] Versions of KEDS prior to 0.9B3 used a system of designating "discard codes," which involved the use of special codes - set using the CODE command in the *.Options* file - whose text started with ~ ~ . If you are using an older dictionary these may still be present; the relevant phrases should be moved to the <discard> class statement. In version 0.9, these codes are automatically converted to a standard discard code, but this feature will not be maintained indefinitely.

Soviet Union, or to a violence-prone region afflicted by ethnic conflict and political demagogues in the southern region of the United States. To a human coder, the choice of the two GEORGIAs is usually clear from the context of the text, but it may be difficult to distinguish the cases using only the sparse parsing available in KEDS. Diverting the cases to the *.complex* file allows these to be later coded by a human. In some coding systems, the complex code might be used with verb phrases whose coding almost always require information beyond that available in the SVO structure.

5.5 Special Purpose Codes for Actors

5.5.1 Coded Compound Actors

In some circumstances, it is useful to have a single phrase generate multiple actor codes. This is referred to as a *coded* compound actor, as distinct from a *parsed* compound, which is based on the structure of the sentence.

Coded compound actors are entered by separating the actor codes with a slash ("/"). For example

EAST_AND_WEST_GERMANY [**GME/GMW**]
NORTH_AND_SOUTH_KOREA [**KON/KOS**]

This method can also be used to expand the membership of alliances when that is appropriate:

G7 [**USA/GMW/FRN/ITL/UK/JAP/CAN**]

5.5.2 Date Restricted Codes

In a data set covering a long time period, some individuals will change their role. For example, Boutros Boutros Ghali was in the foreign ministry of Egypt prior to 1 January 1992, then became Secretary General of the United Nations on that date. To code the period 1990-1994, Boutros Ghali needs to be assigned the code EGY for part of the period, and UNO for the remainder.

This problem is handled by assigning multiple codes, and then putting a date restriction on each code. Date restrictions have the following formats; each restricted code goes in its own set of brackets.

<**YYMMDD** Assign the code for events prior to and including this date

><**YYMMDD** Assign the code for events after and including this date

YYMMDD-YYMMDD Assign the code for events between the two dates

Using this formats, the coding for Boutros-Ghali would be

BOUTROS-GHALI [EGY <911231] [UNO >920101]

A code with no date restriction will be used as a default. If Boutros-Ghali has a UN position until 31 December 1999, then returns to a position in the Egyptian government, the following coding would work:

BOUTROS-GHALI [EGY] [UNO 920101-991231]

The codes used with date restrictions can be complex (compound codes and implicit agents) as will as simple codes.

Notes on date restrictions:

1. Years 00 through 10 are assumed to be 2000 through 2010, so in the KEDS coding system the year "01" is greater than "99." If you are still using KEDS after 2010, get a better program...

2. If a date cannot be interpreted (e.g. is less than six characters; contains non-numeric characters or characters out of range), the following defaults are used

Year	14
Month	01
Day	01

   When the date is written by the system (e.g. when the *.Actors* file is saved), a comment will be added to any code containing "**140101**" indicating that it is a possible error.

3. The total length of the codes and date restrictions must be less than or equal to 80 characters; any codes beyond this length will be ignored.

4. If there are inconsistent restrictions, for example

   [XXX <920101] [YYY <930101]

   the first restriction satisfied will be applied. Restrictions are evaluated in the order they are listed, except for the default (unrestricted) code, which is applied only if none of the restrictions are satisfied. If none of the restrictions are satisfied and there is no default code, the null code is assigned.

5. If only a single date restriction is used, e.g.

   EAST GERMANY [GME <901103]

   the system will automatically add a null code as the default:

   EAST GERMANY [GME <901103] [- - -]

These null codes will appear in the modify dialog and in the *.Actors* file when it is saved

5.6 Special Purpose Codes for Verbs

5.6.1 Subordinate Codes

A subordinate code is used only if no other events are found. It is denoted by adding ? to the end of the event code, e.g.

SAID [023?]

The main use for this is coding attribution. Typically Reuters events start with

<actor$_1$> SAID <pronoun> <verb> <actor$_2$>

For example

`GEORGE BUSH SAID HE REJECTED SYRIA'S ASSERTION...`

The relevant event is

USA REJECTED SYRIA

rather than

USA SAID SYRIA

The combination of dereferencing the pronoun HE and using a subordinate code will handle this.

> **Note:** Subordinate events are promoted – converted to regular events – according to the following rules:
>
> 1. Look for a regular event with a valid source and target. If this exists, do not code the subordinate event. "Valid" means the source and target meet the criteria set in the **VALID** command or Options/Valid Event menu option.
> 2. If every event is subordinate, promote the first subordinate event that has a valid source and valid target.

5.6.2 Dominant Codes

A dominant code stops the further coding of events and becomes the only event reported in a sentence or clause;[2] it is used in situations where the coding

[2]Depending on the **CODE BY...** parameter, described below

emphasizes one type of action over any others found in the sentence. A dominant code is specified by adding ! to the end of the code; for example:

REJECT
- * EFFORT BY [111!]

Unlike regular events, phrase with a dominant code does not require a valid source or target to be coded. If you only want the phrase to be dominant if it has a source and target, specify these explicitly in the phrase:

REJECT
- $ * EFFORT BY + [111!]

REFUSED
- $ * + [111!]

In most cases, the command **SET: CHECK DOMINANT = TRUE** needs to be in the *.Options* file if dominant codes are used. This command forces the system to continue to check all of the verbs in a sentence or clause after it has found an event; this insures that if a verb or phrase containing a dominant code is anywhere in the sentence it will be found. Under the default condition, **CHECK DOMINANT = FALSE**, the system stops checking verbs in a sentence or clause once it has found a non-subordinate event. Checking all of the verbs for dominant codes may noticeably slow the program, particularly if verbs in the text are many and dominant codes are few.

Example:

If you wanted to assign a special code to statements that contained a form of an attribution indicating that the source was unreliable, while still recording who was spreading gossip about whom, the verb list might contain

ALLEGED [028!]
RUMORED [028!]
SPECULATED [028!]
UNCONFIRMED_REPORTS [028!]

along with the **SET: CHECK DOMINANT = TRUE** command in the *.Options* file.

5.6.3 Paired Codes

There are an assortment of circumstances where the WEIS coding scheme generates symmetric events of the form

<Actor$_1$> Event$_1$ <Actor$_2$>
<Actor$_2$> Event$_2$ <Actor$_1$>

For example, a meeting between Israel and Egypt would generate the pair

ISR 031 UAR (meet with)
UAR 031 ISR (meet with)

A visit by a Jordanian official to Syria would generate the pair

JOR 032 SYR (visit; go to)
SYR 033 JOR (receive visit; host)

These combinations can be coded automatically by using a pair of codes separated by a colon (:); for example

FLEW
- $ * TO + [032:033]

would do the visit-and-receive pair, while

MEET [031:031]

generates two events with the same code but with the source and target reversed.

> **Note:** In our tests comparing KEDS to human coders, the program's consistency in correctly assigning multiple codes goes a long way towards compensating for its inability to interpret complex sentences. Human coders tend to miss some of the source-target combinations implied by paired codes and compound actors; KEDS gets them all. In Reuters leads, these situations generate a *lot* of events.

Chapter 6

PARSING: HOW KEDS LOOKS AT A SENTENCE

6.1 Purpose

In order to code event data, KEDS must parse a sentence to identify the subject, verb phrase, and object of the sentence. To do this, KEDS employs a number of simple but general grammatical rules for parsing English sentences. These are sufficient to handle most of the sentence structures encountered in Reuters. These standard rules can be supplemented by customized transformations specified in KEDS *rules* statements; this chapter will discuss only the standard transformations. KEDS is not a general purpose natural language parser:[1] if the source text uses complex sentence structures – as might be encountered, for example, in political speeches or legal documents – then KEDS is probably not the appropriate coding program.

KEDS recognizes the following types of words, which are called "classes" in the KEDS system. Some of these correspond to conventional parts of speech (e.g. pronouns, conjunctions), while others are more specific (e.g. actors and agents).

[1] See Pinker (1994) for a wide variety of examples of why machine-parsing is a more difficult problem than it first appears to be. The problem of creating a general machine parser, even for a single language such as English, has defied the efforts of linguists and computer scientists for forty years, and KEDS is no breakthrough in this regard. Instead, the approach taken by KEDS is to use a small number of fairly robust parsing techniques that provide sufficient information about a sentence from a newswire reports that event data can be correctly coded most of the time.

Class	Content
actor	Nouns in the *.Actors* dictionary
agents	Improper nouns, set in *.Options* file
im.actr	Implicit actor: an agent that is treated as an actor
verb	Verbs in the *.Verbs* dictionary (but not words inside verb phrases)
conj	Conjunctions: AND_ and BUT_
pronoun	Pronouns: HE_ SHE_ IT_ THEY_, THEM_ and ITS_
prep	Prepositions: these are set in the *.Options* file and used only in the **Places** option
comma	Commas
stop	Articles: A_ AN_ THE_ (these are discarded)
null	Null-coded actors, null-coded verbs without phrases, and any word not in the dictionary

All of these classes except "null" can be used in the standard parsing; they can also be used to construct complex patterns and rules. In its standard parsing, KEDS does not recognize adjectives, and therefore does not formally recognize noun phrases. This has little impact on coding accuracy for Reuters lead sentences, though it might be important for other texts. If such information is important in classifying, it should be incorporated into a verb phrase.

The standard KEDS parsing does the following:

- Identifies compound actors
- Reduces titles to a single actor reference
- Identifies compound clauses within a sentence
- Locates the references of pronouns
- Ignores stop words
- Eliminates comma-delimited subordinate clauses

Additional parsing features can be activated by using commands in the *.Options* file:

- Association of agents with actors
- Evaluation of prepositional phrases for determining location

6.2 Parsed Compound Actors

KEDS recognizes compound nouns of the form

<np>_AND_<actor>
<np$_1$ >_,_<np$_2$ >_,_...._<np$_{n-1}$ >_AND_<actor$_n$ >

where

np = <actor> | <np>_<agent>[2]

and does the appropriate duplication of events. This is a *parsed* compound actor, as distinct from the *coded* compound discussed earlier. A compound actor is treated as having the regular actor-class during the coding (e.g. during the search for sources and targets).

Compound actors are concatenated in the Word List and Parsed Text Display, with the symbol "&" replacing the word "AND".

The actor must follow *immediately* after the 'AND': the phrase

```
YITZHAK SHAMIR AND A TIRED JAMES BAKER
```

would not code as a compound because of the intervening adjective TIRED. Such constructions are infrequent in Reuters. Words in the <stop> class are an exception to this; these are discussed below.

Example:

The sentence

```
THE UNITED STATES AND EGYPT APPROVED OF EFFORTS BY ISRAEL AND JORDAN
TO ...
```

would code to

```
USA <APPROVED> ISR
USA <APPROVED> JOR
UAR <APPROVED> ISR
UAR <APPROVED> JOR
```

With agent coding and the verb pattern

[2] In other words, the compounds of a compound phrase can be either actors or actors followed by any number of agents.

CLASH
-% * [221:221]

the sentence

```
PALESTINIAN POLICE AND ISRAELI SETTLERS CLASHED...
```

would code to

```
PAL POL <221> ISR CIV
ISR CIV <221> PAL POL
```

> **Note:** The system will not detect a parsed compound actor in combination with a coded compound actor containing a conjunction.[3] For example, if **EAST_AND_WEST_GERMANY** was a coded compound, then the phrase
>
> ```
> POLAND, HUNGARY, EAST AND WEST GERMANY AGREED TO COORDINATE
> ```
>
> would not correctly identify the compound subject because the AND_ is absorbed in the coded compound. To get around this problem, use a word processor or **REPLACE:** command (described in the *.Options* chapter) to replace **EAST AND WEST GERMANY** with **EAST GERMANY AND WEST GERMANY**.

6.2.1 Stop Words in Compounds

In text analysis, the term "stop words" refers to common words that either contribute little to the interpretation of a text, or would require special treatment by the parser because they could be erroneously interpreted. In KEDS, most words and phrases that can cause erroneous interpretation are eliminated using the *null codes* discussed above.

KEDS contains a <stop> class that is used to eliminate articles in compound phrases (and in any other syntactic processing), so that

```
ISRAEL AND THE UNITED STATES DISCUSSED...
```

is converted to

ISRAEL_&_UNITED_STATES

The only words KEDS classifies automatically as stop words are the articles:

A_ AN_ THE_

[3] This is not done as a general parsing rule because coded compounds – e.g. G7, MAJOR_POWERS, GANG_OF_FOUR – do not necessarily contain embedded conjunctions.

Stop words are ignored in the *syntactic* phase of the analysis, but they are still used in the *pattern matching* for verb phrases or in the string matching in ISSUES facility.

Additional stop words can be added with the command

STOP: <text>

in the *.Options* file or by using <**stop**> = in the *.Classes* file. It is advisable to use an underscore at the end of a stop word to guarantee that no stemming will be done: for example using the stop word

STOP: OF

would eliminate such words as `OFFICIAL`, `OFFERED`, and `OFTEN`, whereas

STOP: OF_

only eliminates `OF`.

6.3 Titles

With an appropriately general dictionary of actors, a phrases such as

```
Israeli Prime Minister Yitzhak Shamir
```

will result in two <actor> entries: one for `Israeli` and one for `Shamir`. This can be eliminated by designating the phrase **PRIME_MINISTER** as a <title>: When a <title> is found, the system converts a phrase of the form.

<actor$_1$ > <title> <actor$_2$ >

into a single <actor> if and only if <actor$_1$ > and <actor$_2$ > refer to the same code. <title> does not have to be an official title; for example the phrase

```
Palestinian leader Yasar Arafat
```

could be converted to a single actor by designating the word leader as a title. Titles are set in the *.Options* file or by using a **CLASSES:** commands.

Example

The phrase

```
Israeli Prime Minister Yitzhak Shamir and Egyptian President Hosni
Mubarak have agreed to hold talks soon, Israel Radio said
```

would code to

ISR ISR_AND_UAR UAR

in the absence of titles. With the *.Options* file commands

TITLE: PRIME_MINISTER
TITLE: PRESIDENT

the phrase codes to

ISR_AND_UAR

and is parsed as a compound actor.

> **Note:** Because KEDS automatically searches for a target that has a code distinct from that of the source, a complete list of titles is usually not critical for coding accuracy, but it may be useful in some cases.

6.4 Compound Sentences

By default, if a sentence is compound – that is, it contains multiple clauses separated with conjunctions – then each clause will be coded, so the sentence could generate multiple events. This can be changed using the **SET: CODE BY ...** command discussed below. The first event in a clauses that is recognized will be coded (with adjustments for subordinate and dominant codes). This means events are prioritized by:

- left to right order of verbs in the clause
- phrases within a verb by length

The two standard conjunctions are **AND_** and **BUT_**, though additional conjunctions can be added using a *.Classes* file. ANDs that are used in a compound actor formation or by a verb phrase that itself contains a compound (e.g. **SHOT_AND_KILLED**) are not used to delimit clauses.

When a compound sentence is encountered, the source actor is kept the same unless it is explicitly overridden by a $ operator in a phrase or if an actor occurs *immediately* after the conjunction. In coding after a conjunction, the target is reset using the default rules for assigning targets (e.g. the code of the target must be distinct from the code of the source).

If no event is found in the first clause, the source actor is set to the first actor in that clause, provided one exists. This usually identifies the correct subject of a sentence even when the verb in the first clause is not in the dictionary.

Note: A maximum of 8 conjunctions are tracked for purposes delimiting clauses. If there are more conjunctions in the sentence, the remainder of the sentence is treated as a single clause; such cases are quite rare.

6.4.1 CODE BY... Command

The **SET: CODE BY ...** parameter, set in the *.Options* file, determines when the system stops looking for events. There are three options:

CODE BY SENTENCE
The system codes only the first event it finds in the sentence. A subordinate event is promoted only if no other event is found in the sentence. If CHECK DOMINANT=TRUE, the first dominant event found in the sentence is coded.

CODE BY CLAUSE.
The system codes the first event it finds in each conjunction-delimited clause in the sentence. A subordinate event is promoted only if no other event is found in the clause containing the subordinate event. If CHECK DOMINANT=TRUE, the first dominant event found in each clause is coded.

CODE ALL.
The system codes every event that it finds in the sentence. A subordinate event is promoted only if no other event is found in the sentence. If CHECK DOMINANT=TRUE, the first dominant event found in the sentence is coded.

6.4.2 Effects of SET: CODE BY on Event Selection

	CHECK DOMINANT= FALSE	CHECK DOMINANT= TRUE
CLAUSE	The first verb phrase in each conjunction-delimited clause that contains a non-subordinate code and a valid source and target is coded.	The first verb phrase in each conjunction-delimited clause that contains a non-subordinate code and a valid source and target is coded. If a verb phrase with a dominant code is found anywhere in a clause, it overrides any other events coded in the clause.
SENTENCE	The first verb phrase in the sentence that contains a non-subordinate code and a valid source and target is coded.	The first verb phrase in the sentence that contains a non-subordinate code and a valid source and target is coded. If a verb phrase with a dominant code is found anywhere in the sentence, it overrides any other events coded in the sentence.
ALL	All verb phrases in the sentence are coded.	All verb phrases in the sentence are coded. If a verb phrase with a dominant code is found anywhere in the sentence, it overrides any other events coded in the sentence.

Notes:

1. If all events found in a sentence are subordinate, one event is promoted according to the rules described in the Subordinate Code section.
2. If a dominant event is found, it is coded even if it contains a missing source or target.
3. The conditions for a valid source and target are set using a VALID command or with the Options/Valid Events menu command.

Example

If **BOMBED, SAID** and **ATTACK** are verbs, then the sentence

```
Israel bombed guerrilla bases in southern Lebanon and said this was
in retaliation for earlier rocket attacks...
```

CODE BY SENTENCE would generate one event

```
ISR BOMBED LEB
```

CODE BY CLAUSE would generate two events

```
ISR BOMBED LEB
ISR SAID LEB
```

CODE ALL would generate three events

```
ISR BOMBED LEB
ISR SAID LEB
ISR ATTACK LEB
```

6.4.3 SET: IGNORE CONJUNCTIONS = TRUE Command

This *.Options* file command causes the pattern matching to ignore conjunctions, so that a match can occur across a conjunction even if the conjunction was not absorbed by a verb root. This usually is not desirable and can have some unanticipated effects. For example the *.Verbs* entries[4]

KILLED [678]
- SHOT_AND_* [123]
. . .
SHOT [345]

and the phrase "XXX shot and killed YYY" will create two separate events:

XXX SHOT [345] YYY
XXX SHOT_AND_KILLED [123] YYY

because the SHOT_AND_* patterns is matched across the conjunction, and SHOT is coded because it is in the first part of a compound sentence. With **IGNORE CONJUNCTIONS=FALSE** the coding is

XXX SHOT [345] *** (no target)
XXX KILLED [678] YYY

[4]This would not be a particularly brilliant combination of verb phrases in the best of circumstances but it is the sort of thing that can occur when multiple, inexperienced coders are working on the dictionaries...

6.5 Dereferencing Pronouns

Pronouns occur fairly frequently in Reuters, for example in the phrase:

```
Turkey believes Iraq and Syria can cope with a decrease in vital water
they receive from the mighty Euphrates river when a major dam is filled
next month.
```

Dereferencing pronouns involves ascertaining which noun or nouns a pronoun refers to; in this example `THEY` refers to `IRAQ AND SYRIA`. Dereferencing is a very general problem in parsing[5] the techniques used in KEDS are quite simple, though fairly effective when applied to Reuters leads. In compound sentences, dereferencing is important in bringing actor references into the appropriate clause of the sentence. The pronoun `IT_` is most likely to be incorrectly dereferenced since `IT_` sometimes refers to an action, an abstract concept, or a direct object.[6]

KEDS uses the following rules to handle pronoun dereferencing:[7]

`HE, SHE, IT` assign the first actor in the sentence

`ITS, HIS, HER` assign to the first actor prior to the pronoun

`THEY, THEM, THEIR` assign either:

- First compound actor if one exists
- First actor followed by a word ending in 'S', a specified plural or an agent[8] (e.g. `POLISH MILITIA, SYRIAN SOLDIERS`)

In the Word List display, dereferenced pronouns are typed as <**actor**> and show up as the pronoun followed by the symbol "– >" followed by the reference. In the example above, `THEY` would appear as

THEY– >IRAQ_&_SYRIA

When full stories, rather than only lead sentences, are being coded, pronoun dereferencing can also occur across source texts; see the discussion of the FORWARD command in the *.OPTIONS* chapter.

[5] It is also a problem that cannot be solved on a purely syntactic basis: in the sentence `BAKER WILL MEET WITH MUBARAK WHEN HE GOES TO GENEVA` the pronoun `HE` could refer to either `BAKER` or `MUBARAK` depending on who is going to Geneva.

[6] Because English syntax require a subject, the interpretation of IT is further complicated by its use as a placeholder in sentences that have *no* subject – `It is raining` – or as the object when a transitive verb – `The ambassadors discussed it over dinner.`

[7] It is *possible* to add additional pronouns using a CLASS command: these will be assigned to the first actor unless they begin with a 'T' (as in THEY and THEM) or have 'S' as the third letter (as in ITS).

[8] See comments on plurals and agents in the *.Options* File and Agents chapters

6.6 Elimination of Comma-Delimited Nonrestrictive Clauses

In Reuters leads, short clauses delimited by commas and clauses between commas and the end of the sentence are usually irrelevant to coding events:

```
President Hosni Mubarak, in a grim warning underlining Egypt's deepening
economic crisis, will request emergency assistance from the International
Monetary Fund, the official UAE news agency WAM said on Thursday ..
```

Following the vocabulary used in Turabian (1987:51), these are called *nonrestrictive clauses*, or *comma-delimited clauses*.[9]

KEDS's default action is to eliminate these phrases if the number of words between commas is greater than two and less than or equal to ten; the minimum allows the preservation of lists with commas in them. The maximum and minimum length of an eliminated phrase can be adjusted using the **COMMA** command in the *.Options* file; this feature can also be turned off. Commas inside numbers – for example 10,000 – do not trigger this feature, nor do commas insides lists of actors that were converted to parsed compounds.

If the **SET: CODE NONRESTRICTIVE =TRUE** command is used in the *.Options* file, the text inside the nonrestrictive clauses is checked for events if and only if no events were found in the text that remained after those clauses were deleted. In our experience, this strategy is likely to result in a number of incorrectly coded events, but it might be useful in dealing with material where the editors made liberal use of commas.

See Also: OMIT command

6.7 Summary: Parsing

Parsing operations occur in the following order; operations in *italics* are optional:

1. Filter the text for punctuation, diacriticals and REPLACE rules.
2. Assign classes to all words that are in the dictionary; if a word cannot be found it is classified <null>
3. *Apply rules*

[9] In some earlier versions of KEDS these were called "subordinate clauses," which was an incorrect use of that grammatical term. Some comma-delimited clauses found in event reports are not nonrestrictive by Turabian's definition either – Turabian distinguishes 14 different types of comma-delimited clauses – but this term will do for purposes of machine coding...

4. Reduce title phrases
5. *Assign actors to agents*
6. Reduce compound actors
7. Dereference pronouns
8. Eliminate comma-delimited nonrestrictive clauses
9. Identify the clauses in a compound sentence.
10. Check the phrases for each verb in each clause of the sentence
11. When a verb phrase is found and does not have designated actors, identify source and target:
    (a) Source is the first actor in the sentence
    (b) Target is the first actor after the verb that has a code distinct from that of the source; if this does not exist, it is the first actor before the verb
12. Identify issues using the issues lists.

Chapter 7

.OPTIONS FILE

7.1 Purpose

The *.Options* file is used to provide additional processing information and contains a variety of specific commands.[1] The commands in this file can have a ***substantial*** impact on how KEDS does coding, and a dictionary that has been developed under one set of *.Options* settings may work very poorly under another set.

This chapter discusses some of the general commands in the *Options* file. A number of other commands discussed in later chapters for example formatting, filtering, issues and agents are also activated using commands in the *Options* file.

7.2 .Options Command Notes

1. The *.Options* file should be created using a word processor and saved in flat ASCII (TEXT) format. The file cannot be changed from within the KEDS program, though some of the options can be temporarily reset while the program is running.

2. The command names are given here in full for mnemonic purposes, but the program usually only looks at the first four characters and the colon. For example,

   TARGET: CZE

   could be entered as

[1] In earlier versions of KEDS this was called the "Special" file.

TARG:CZE

3. At the present time, there is little error checking in most of the commands in the *.Options* file. If an improper format is used, the value of the parameter will usually stay at the default but no indication of an error will be given. The commands and parameter names are **case-sensitive**.

4. The *.Options* file can take input lines up to 255 characters in length, and in many applications the ISSUE, DISPLAY and OUTPUT commands may be longer than the 80-character line length used in many words processors. Be sure to save these as TEXT without the line feeds so that the command is read as a single string of characters.

7.3 SET <parameter> = <value>

This is a general command that sets various switches and numerical parameters affecting the operation of the program. Switches are logical and can take the values **TRUE (T)** or **FALSE (F)**; numerical parameters have various formats. The details of most of these settings are given in the sections where their effects are discussed; the LOCK and DEMO settings are discussed below.

List of SET switches and parameters

Parameter	Section	Values	Default
FILTER INPUT	Input	logical	TRUE
CODE AGENTS	Agent	logical	FALSE
AGENT SEARCH = (<back>,<frwd>)	Agent	numerical	(2,2)*
CONVERT AGENTS	Agent	logical	TRUE*
REPLACE EXPLICIT AGENTS	Agent	logical	FALSE*
CODE BY ...	Parsing	CLAUSE ALL SENTENCE	CLAUSE
CODE NONRESTRICTIVE	Parsing	logical	FALSE
CHECK DOMINANT	Special codes	logical	FALSE
CODE PLACES	Options	logical	FALSE
IGNORE CONJUNCTIONS	Compound sentences	logical	FALSE
NOTES	Formatting	logical	FALSE
PANDA FORMAT	Formatting	logical	FALSE
TEXT FILE	Formatting	logical	FALSE
PROBLEMS FILE	Formatting	logical	FALSE
LOCK	Options	logical	FALSE
DEMO	Options	logical	FALSE

* This is active only if the SET:CODE AGENTS=TRUE command has been used

7.3.1 SET: LOCK = TRUE

This command disables all of the **Modify** menu options except for **New Event**. It is designed to be used if you want to prevent changes from being made in the coding

dictionaries when doing human-assisted coding.

7.3.2 SET: DEMO = TRUE

This command is designed to be used when constructing a demonstration file (e.g. **KEDS.Sample**) and disables the following features:

1. Records are not skipped at the beginning of the file, even if some have been coded previously;
2. Changes in the actors, verbs and classes lists are not saved;
3. When the end of the *.Text* file is reached, the program simply terminates rather than asking whether you want to code another file.

7.4 CODE <code string> = <text>

This command assigns a text string to a code. The text can appear in the output strings displayed in the event window of the program and providing some text to describe each event code is strongly recommended when doing machine-assisted coding. Text can be assigned to either actor or event codes.

Examples:

CODE: 04=APPROVE
CODE: 041=PRAISE
CODE: 042=ENDORSE
CODE: 05=PROMISE
CODE: 051=PROMISE POLICY SUPPORT
CODE: ISR=Israel
CODE: PAL=Palestinians
CODE: JOR=Jordan

7.5 REPLACE: '*string1*' WITH '*string2*'

These commands cause an iterated, line-by-line, search-and-replace to be done on the input text before *any* additional processing (including punctuation filtering) is done. Its typical application is to change characters that would otherwise cause problems after filtering (e.g. diacriticals, smart quotes), getting rid of problematic hyphenated clauses and standardizing the spelling of verbs. There can be multiple **REPLACE** commands in the *.Options* file.[2]

[2]The actual command syntax is quite flexible: the system just looks for the character string REPL and two strings delimited with '.' The following are equivalent:
REPLACE 'ABC' WITH 'XYZ'
REPL 'ABC' 'XYZ'
REPLACE 'ABC'– >'XYZ'

A *string* can be anything (including a null string) between two single quotes. To put a single-quote in a string, use two consecutive single quotes. **REPLACE** is applied *before* characters are shifted to upper case, so for example 'Á' and 'á' are distinct characters.

REPLACE is useful for standardizing the spelling of verbs – for example in dealing with American and British spelling – because separate sets of phrases would otherwise need to be maintained for each spelling of the verb. Multiple spellings of an actor (e.g. QADHAFI, QADDAFI, GADDAFI), in contrast, are more efficiently handled by making multiple entries in the *.Actors* file.

The **REPLACE** command is not particularly efficient, and if you are going to be coding a set of texts several times, it will be faster to use a word processing to do the substitutions in the original text file, rather than doing the substitutions during the processing by KEDS. The search-and-replace of a word processor may also be preferable if you want to search for patterns by paragraph (i.e. across the line breaks in the text) rather than by line, or if the replacements could result in an infinite loop (see Note below).

Examples

REPLACE: 'Å' WITH 'AO'
REPLACE: 'AL-' WITH ' '
REPLACE: 'CANNOT' WITH 'CAN'T'
REPLACE: 'ORGANISE' WITH 'ORGANIZE'

> **Note:** Because the **REPLACE** command is iterated, it is possible to create a **REPLACE** that generates an infinite loop; for example

REPLACE: 'ABC' WITH 'ABCZ'

would convert ABCDEF to ABCZZZZZZZZ...ZZZZZZDEF. This is a different behavior than the search-and-replace in most word processors, which usually replace a pattern once and then skip to the *next* occurrence of the target pattern.

Rather than repeat an infinite loop indefinitely, the **REPLACE** command will stop (and beep) after a finite number of substitutions equal to the number of characters in the original source string. The infinite replacement problem will usually be evident in the displayed text, though the string will have been truncated to 96 characters and some of the extensions may be lost. This will also be evident from the programming beeping during autocoding.

7.6 FORWARD: sequence_format FORWARD: THEY

The **FORWARD** command activates the forwarding of pronoun references between text records: this is used when coding full stories. When **FORWARD** is active, if a pronoun occurs at the beginning of a sentence, prior to any actors, it is dereferenced

as the first actor in the *previous* sentence, where "previous" is determined by the sequence format given in the command, provided that actor occurred before a verb.

By default, the plural pronouns THEY_, THEM_ and THEIR_ are only replaced if the forwarded actor is compound. This can be deactivated using the **FORWARD: THEY** command described below.

The sequencing of sentences is given by the *idinfo* string in the date header. The sequence_format string in the command (discussed below) shows where the sequencing information is found. Forwarding is done if the following conditions hold:

- The story number of the current text is *equal* to the story number of the previous source text
- The sentence number of the current text is equal to the sentence number of the previous source text *plus one*

Note that the second condition allows your formatting program to specify situations where forwarding should not be done (e.g. across paragraphs) by increasing the sentence number more than 1. Story and sentence numbers must be less than or equal to 32,767.

Pronouns can be forwarded across multiple stories provided a chains of references is maintained. For example, in the sequence

```
810726 REUT-045-001
James Baker begins a Middle East shuttle trip...

810726 REUT-045-002
He will first fly to Israel to...

810726 REUT-045-003
Afterwards, he plans to visit Jordan to...
```

The "he" in the third sentence (45-003) will still refer to Baker.

7.6.1 Formatting the Sequence Numbers

The sequence_format string is composed of the following characters:

^ = leading blanks between the date and the start of the idinfo string

* = any character (including blanks after the start of the string)

N = serial number of the story

S = serial number of the sentence within the story

For example, if your text formater had generated story id's consisting of a 3-character source identifier, a 3-digit story identifier and a 3-digit sentence identifier:

```
890227REU312023
```

the serial format would be

*****NNNSSS**

If this format included some blanks and hyphens for readability

```
890227◇◇REU-312-023
```

(where ◇=blank) the serial format would be

```
^^****NNN*SSS
```

when the FORWARD command is used, the *idinfo* string is copied for the first non-blank (i.e. *, N or S) character to the end of the string, so in the first example the idinfo string would be nine (9) characters in length; in the second example it would be eleven (11) characters in length. The maximum length of an idinfo string – not including leading blanks – is 15 characters.

The story and sentence serial numbers must consist of a sub-string of consecutive N's and S's (i.e. ***NNSSNN is not allowed; the second set of NN will be ignored) but can be in any order (i.e. SSS***NNN is allowed). If a non-numeric value is found in either sequence number, the entire number is treated as equal to zero.

> **Note:** If a sentence sequence number (SSS) is not included, the FORWARD command has the effect of extending the length of the idinfo string beyond the default length of 6 without ever doing pronoun forwarding.

7.6.2 Forwarding plural pronouns

The plural pronouns THEY_, THEM_ and THEIR_ are only replaced if the forwarded actor is compound. These rules, while somewhat conservative, is quite effective at handling pronoun references across sentences in Reuters. The additional command

FORWARD: THEY

deactivates this rule, so that plural pronouns obey the same rules as singular pronouns. This is useful if you have a number of actors (e.g. political parties) who are plural.[3] This command can occur anywhere in the file; it does not need to follow the original **FORWARD** command.

[3] At the present time there is no simple way to designate an actor as being a plural; a future version of KEDS will probably correct this. The PLURAL command only works with common nouns (although intelligent use of PLURAL can take care of many cases). A complicated way to do this would involve assigning a bogus compound code
OPPOSITION_PARTIES [ALBOPP/XXX]
and then filtering out all of the records with XXX as actor or target.

7.7 SOURCE: pattern | [code]
TARGET: pattern | [code]

The **SOURCE** and **TARGET** commands provide additional default search *patterns* that look for actors associated with specific words, or provide a default *code* for the source or target. These are used only if the default search for actors before and after the verb has failed.

Examples:
SOURCE: $ SAID
TARGET: WITH +
TARGET: TO +

A **SOURCE** or **TARGET** pattern must contain a $ or + token. Multiple patterns can be specified using multiple **SOURCE/TARGET** commands, and these are searched in the following order:

1. All actors in the sentence are checked in left to right order. The actor will anchor the pattern match at the location of the $ or + token.
2. Using this anchor, the patterns are checked in order of length, with the longest pattern checked first.
3. Unless the **SET:IGNORE CONJUNCTIONS=TRUE** option has been used, the pattern must match within the conjunction-delimited clause containing the actor.

7.7.1 Default Codes

If a bracketed *code* is specified after **SOURCE** or **TARGET**,

TARGET: [WORLD]
SOURCE: [USA:GOV][4]

it will be used to generate a default source or target code even when no identifiable actor or agent can be found associated with the verb. Effectively, the default code replaces the missing code ***. This is useful in two circumstances:

- It will sometimes allow the coding of abbreviated, syntactically incomplete sentencesfor example chronologies or direct quoteswhere the subject or object is implied rather than explicit.[5]
- When you are using KEDS to code only verb phrases and are not interested in sources and targets.

Default codes that refer to specific actors – as opposed to special codes that are only used in the context of a default – will probably substantially increase the number of

[4] This example includes an implicit agent in the code.
[5] See the discussion in Gerner, et al. 1994.

incorrectly coded events; it is generally a good idea to make the default codes distinct (e.g. SRCDEF, TARDEF) so that you know they have been assigned by the default procedure rather than through regular coding. As noted earlier, assigning a default source will guarantee that the first verb in each clause (or the sentence) is coded. English words that look like verbs may actually be nouns or adjectives so coding that uses a default source is less likely to identify the actual verb in a clause or sentence.

7.8 PREP, TITLE and PLURAL

As discussed in the Parsing chapter, the words in the <prep>, <title> and <plural> classes are used by the KEDS parser. Words in the <plural> class affect the dereferencing of the pronoun `THEY`, words in the <prep> class are used to determine the location of events when the PLACE facility is used, and <title> are used to reduce two actor references to a single reference. These have the same format:

PLURAL: string
PREP: string
TITLE: string

Unlike the lists in the *.Classes* file, these commands have only one phrase per command. These commands are equivalent to setting the words using a <**class**>= command n the *.Classes* file (and are a vestige from versions of KEDS that did not have the CLASSES command) but may occasionally allow the *.Classes* file to be skipped.

Example
PREP: IN_
PREP: IN_THE
PREP: FROM_
TITLE: PRIME_MINISTER
TITLE: PRESIDENT
PLURAL: MILITIA
PLURAL: POLICE

> **Note:** These can also be defined using the *.Classes* file, and they are saved in the appropriate class statement whenever the *.Classes* file is saved when exiting the program. Because of this, if you are using both these statements and a *.Classes* file, you'll get a series of alert messages indicating that the word has already been assigned a type. The way around this is to just make the assignments in the *.Classes* file if you are using classes.

7.9 COMMA

This command sets the minimum and maximum length of comma-delimited nonrestrictive clauses that are eliminated before coding. The syntax is

COMMA: MIN=<minimum value> MAX=<maximum value>
{EMIN=<minimum value> EMAX=<maximum value>}
{BMIN=<minimum value> BMAX=<maximum value>}

where the values are the number of words between commas, or between a comma and the end of a phrase.

The BMIN/BMAX limits are used only on phrases at the beginning of the sentence and the EMIN/EMAX limits are used only on phrases at the end of the sentence. If BMIN/BMAX and EMIN/EMAX are not specificed, EMIN/EMAX are set to the same values as MIN/MAX and BMIN/BMAX are set to their defaults (i.e. inactive).

The defaults are

BMIN =	**255**	**BMAX =**	**0**
MIN =	**2**	**MAX =**	**2**
EMIN =	**2**	**EMAX =**	**10**

The default BMIN/BMAX settings cause the first comma-delimited phrase to **never** to deleted.

This feature can be turned off for all phrases using the command

COMMA: OFF

When this command is used, all of the text – including the material in comma-delimited clauses – is checked for events. This will usually substantially change the coding. To code from the nonrestrictive clauses only when no events were found in the regular text, use the **SET: CODE NONRESTRICTIVE=TRUE** command.

Example
COMMA: MIN = 1 MAX = 8 EMIN=3 EMAX=10
COMMA: OFF

> **Note:** Setting BMIN/BMAX to an active combination such as the MIN/MAX defaults [2,10] will often result in most of the sentence not being coded.

7.10 OMIT

The default delimiters for indicating text that should not be coded are **/*...*/**; these are similar to the "comment" delimiters used in programming languages. Up to three additional sets of omit delimiters can be specified using the **OMIT** command. The syntax of the command is **OMIT** followed by the two delimiters, separated by at least one space.

OMIT: [◇]
OMIT: (*◇*)

(◇ = the blank character)

OMIT delimiters can contain up to three characters each.

The typical use of **OMIT** delimiters is to excise non-language material that shouldn't be coded. For example, the event chronologies in the *Journal of Palestine Studies* have square brackets around information concerning the original source of the event; by designating [ ] as delimiters, this information will automatically be excluded

```
/* To prevent demonstrations commemorating Fateh's
first operation,*/ Israel imposes indefinite
curfew on Occupied Territories [WP, LAT 1/2].
```

In human-assisted coding, adding delimiters allows parts of a sentence that will cause errors when coded to remain in the text for purposes of reference without affecting the coding. This is useful if one will be experimenting with multiple coding schemes and therefore going through the same set of texts on multiple occasions. In the example above, the second clause of the text contains the event; by eliminating the first clause and the comma, the remaining text will be coded correctly.

> **Note:** If an opening omit delimiter is not followed by a closing delimiter, no text following the opening delimiter will be coded. This feature may not be maintained in future versions of the program, so please balance the delimiters – each opening delimiter should have a corresponding closing delimiter. A closing delimiter occurring *before* an opening delimiter has no effect on the coding.

7.11 SAVE

KEDS has an automatic save feature that prompts you to save the files after a specific numbers of changes to the verbs, actors and classes/rules lists. (You still have the option of not saving them despite the program prompting you). The default number of changes is **16** but it can be changed using the **SAVE** command. The syntax is

SAVE: VERBS=<value> ACTORS=<value>
RULES=<value>

where the values are the number of changes (add, change or delete) between prompts. Each part of the command is optional; if it is not present the default value will be used.

The command

SAVE: OFF

turns off the feature.[6]

[6]Strictly speaking, OFF doesn't actually turn off the feature but sets the parameters at a value of 64: Knowing the possibility of unexpected crashes of the operating system (or even KEDS – it can happen even now!), to say nothing of the effects of power failures, in good conscience we can't let anyone enter that many changes without at least reminding them to do a save...

Example
SAVE: VERBS = 12 RULES = 4

7.12 Complexity Filter

The **COMPLEX** command defines the conditions under which a sentence is considered too complex to code. By default all options are inactive except DISCARD and COMPLEX. When a complex text is found, the program takes the following actions:

In autocoding mode: By default, the text is diverted to a *.complex* file that can be later coded using machine-assisted human coding. If the NOSAVE option is used, the record is skipped and no file is written.

In normal mode: No events are coded and the message **Complex text** is displayed. The **Modify/New Event** menu option can be used to create events.

The COMPLEX command has the following options:

Parameter	Condition
VERBS[n]	Diverted if text contains *n* or more verbs
ACTORS[n]	Diverted if text contains *n* or more actors
PRONOUNS[n]	Diverted if text contains *n* or more pronouns
CONJ[n]	Diverted if text contains *n* or more conjunctions
COMMA[n]	Diverted if text contains *n* or more comma-delimited nonrestrictive clauses
LATEVERB[n]	Diverted if there no verb is found within *n* words of the beginning of the sentence. If **LATEVERB** is active, a text will also be diverted if it contains no verb.
COMPLEX[n]	Diverted if text contains *n* or more phrases with complex codes [+++]. Default = 1.
NOACTPRIOR	Diverted if no actor occurs *prior* to the first verb
NOACTAFTER	Diverted if no actor occurs *after* the first verb
NOVERB	Diverted if there is no verb
NOSOURCE	Diverted if there is no source
NOTARGET	Diverted if there is no target
NOEVENT	Diverted if there is no event
EXPLAIN	Inserts an explanation of why the source was diverted into the output file inside /*...*/ delimiters; the program also displays this explanation on the screen after the **Complex text** message when not autocoding
NOSAVE	Do not save the complex files to a *.complex* file when autocoding.
DISCARD[n]	Do not code text if contains *n* or more phrases with discard codes [# # #]. Default = 1. Unlike COMPLEX[n], this option **does not** divert text to the *.complex* file.

The text is diverted if any of the conditions listed in the **COMPLEX** command are met. The first six commands require a number in the brackets; there are no default values, and a value of zero indicates that the condition is not active. The last seven are Boolean; if they are present the condition is checked; otherwise it isn't. As noted above, text will also be diverted to the *.complex* file (or skipped if the NOSAVE option is used) if it contains a phrase that has a complex code +++

Example
COMPLEX: VERBS[5] ACTORS[8] LATEVERB[8] NOVERB NOACTPRIOR EXPLAIN

> **Note:** Null-coded actors and verbs that are null-coded and have no pattern are converted to type NULL and are not considered in the evaluation of complexity.

7.12.1 Setting Complexity Conditions using Rules

Rules can be used to determine complexity by combining a rule with a string that is assigned the complex code. For example, this construction:

In *.Classes*
~ {KING|SADDAM} HUSSEIN ~ {JORDAN|IRAQ} -> XELPMOC

In *.Actors*
XELPMOC [+++]

will divert to the .complex file any text containing a reference to HUSSEIN that does not also contain a reference to KING/SADDAM or JORDAN/IRAQ. This allows complicated definitions of what constitutes a case that meets the complexity criterion.

7.13 PAUSE

This command re-initializes the options in the **Pause When** menu option:[7]

PAUSE:<ALWAYS | CONTAINS | MISSING>
<SOURCE><TARGET><EVENT>
<COMPLEX>

The **PAUSE** command sets the conditions under which KEDS pauses before coding the next record, giving you the option of recoding. This is useful is one is dealing with source material that contains irrelevant material such as purely internal events, when you only want to look at the text when KEDS has not found any events, or when you only want to stop to code complex text.

[7] If this command is not used, the options remain at whatever settings were during the previous coding session. This command – and **VALID** – are useful if you want to be sure that the options are at the values you want them to be even if someone else has been using the project file.

ALWAYS, CONTAINS and **MISSING** control pausing based on the source-event-target coding. The three possibilities are:

Always

KEDS pauses after coding each event; this is the default.

Contains

Pause when the coded text contains *all* of the listed items **Source, Target,** and **Event**. This option is typically used for filtering in machine assisted coding to bypass sentences that contain nothing of interest. Note that if you are only interested in verb phrases ("events") and not actors (for example if you are using KEDS to code something other than event data...), sources and targets can be set to default values.

Missing

Pause if the coded text is missing *any* of the listed items. This option is typically used in machine assisted coding to fill in missing information.

> **Note: Contains** acts as a logical AND while **Missing** acts as a logical OR in this selection. If the input contains a discard phrase, it is always skipped if the **Contains** or **Missing** options are active.

Complex causes the program to pause whenever complex text is found; this acts as a logical OR with the code-related options.

Example
PAUSE: CONTAINS SOURCE EVENT
PAUSE: MISSING TARGET COMPLEX

7.14 VALID

This command re-initializes the options in the Valid Event menu option:[8]

VALID: <SOURCE><TARGET><DISPLAY>

VALID sets the conditions for whether an event is considered valid and written to the *.events* file. The default condition requires both a source and a target, but either of these requirements can be changed using this command (or using the Options/VALID menu option); the default condition also shows invalid events.

The validity condition is true if an event has an non-null code for the actor (source or target). If agents are being coded, either the actor or the agent must be non-null.

If DISPLAY is present in this command, the DISPLAY/Show Invalid Events menu option is checked (active), otherwise it is not checked.

[8] As with PAUSE, these setting are preserved from the previous coding session if this command is not used.

Example

VALID: TARGET DISPLAY	Valid events must have a target and may or may not have a source.[9] Show Invalid Events is checked so invalid events will be displayed.
VALID: SOURCE TARGET	Valid events must have a source and target. Invalid events will not be displayed.
VALID: SOURCE	Valid event must have a source and may or may not have a target. Invalid events will not be displayed.
VALID:	Neither a source nor target is required for a valid event.

7.15 AUTOLOG: 'file_name'

The **AUTOLOG** command specifies a file that will be used to maintain a record of all of the autocoding sessions: this is useful if you are coding a number of different text files and need to keep track of when and how the various files were coded.

The autolog file is of type text and tab-delimited, so it can easily be read into a spreadsheet or database program. It records the following information

- date of the autocoding session
- time the autocoding session was finished
- *.text* file coded
- number of records coded since the *.text* file was opened[10]
- number of events coded since the *.text* file was opened
- time required for coding
- coder ID
- *.project, .verb, .actors, .classes* and *.options* files used

The autocode file is cumulative: results from the current autocode are appended to those of earlier coding sessions. The autocode file does not have to be unique to the project: different projects can use the same file.

Example

AUTOLOG: 'REUTER80-95.AUTO'

[10] In other words, the "Records coded" statistic from the events window. If any manual coding or other autocoding sessions were done earlier, this will be included in this statistic and the "number of events" statistic. The Autolog file is designed to record the autocoding of entire files, where this is not an issue.

7.16 VERIFY: format string

The **VERIFY** command was developed to assist in the debugging of KEDS but might be of use in maintaining the consistency of dictionaries over time. The command does an automatic comparison of the actual codes to expected codes and pauses whenever a difference is detected.

To do a verfiy, specify the codes that you are interested in checking in a format string (described below in the Formatting Output chapter) ; the code fields must be blank delimited.

Each record that is to be verified should be followed by "verify strings" with the following format

1. First character in the line is { and a } occurs after the final code. Anything after the } will be ignored; this can be used for comments;
2. Coding fields are / (slash) delimited. Consecutive slashes (e.g. //) mean that the field should be ignored in this record
3. {~ means that all subsequent verify strings should be ignored
4. There should be as many verify strings as the event records you expect to find (unless you use {~ } to bypass records) and the verify strings should be in the same order as the events.

When the VERIFY option is active, the program will pause whenever there is a discrepancy between the verify string and the coded events. Blanks are trimmed from the code strings before the comparison is made.

Example[11]

```
VERIFY:SOURCE  TARGET  EVENT[3]

890401 PATMATCH
Thirteen Palestinians were test1 edf anchored by Israel
and arrived in Syria on Sunday next ^/
{PAL/ISR/PA2} should *not* code PA5
{SYR/***/PB1}
```

[11] Yes, this is what KEDS verification source texts look like...

Chapter 8

CLASSES, COMPOSITE PATTERNS, AND RULES

8.1 Purpose

KEDS was originally designed as a simple parser working with the subject-verb-object (SVO) structure of English sentences, a small number of grammatical rules and a large number of patterns. As we accumulated experience with the program we encountered a number of situations where a sentence was almost, but not quite, in SVO form, and could be converted to the SVO form by the application of a general rule. We also noticed that general rules could be used to resolve some situations of agent assignment and conjunctions.

While some grammatical rules have been built into the program, KEDS also allows new rules to be specified using two formal grammatical structures: regular patterns and a transformational rules.[1] These facilities are embedded in three features of KEDS:

Classes: These are sets of words that can be used in patterns. These operate in a

[1] In linguistics, the term "transformational" has changed over time. I'm using the term in the sense that it was used when Chomsky originally defined categories of grammars: a rule that allows multiple tokens on both sides of a replacement operator. As such a KEDS rule is a step more elaborate than the context-free grammars usually encountered in the syntax of computer languages (e.g. Backus-Naur form; Prolog), which only allow multiple tokens on the right-hand-side of the substitution. Context-free grammars, while more familiar in computer science, are insufficient to handle many of the common transformations found in natural language, such as the reversal of subject and object in English passive-voice.

If you are accustomed to working with grammatical specifications, note that the KEDS rule notation is the reverse of that usually found in grammars: the right-hand-side replaces the left-hand-side rather than the left-hand-side replacing the right-hand-side in notations such as "LHS::= $RHS_1|RHS_2$." The LHS− >RHS notation follows the ordering of the typical search-and-replace command found in software and has proven more intuitive for the average user of KEDS, who tends to be neither a linguist nor a computer scientist.

fashion similar to verbs, actors, agents, pronouns and other word types recognized by KEDS. Classes typically consist of sets of words that are equivalent either from the standpoint of event coding or in parsing.

Composite patterns: These allow alternative options in a pattern match, as well as patterns within patterns.[2]

Rules: These allow one pattern to be replaced with another pattern.

Classes, composite patterns, and rules are the "bells and whistles" of KEDS – the program works fine (and in fact did for several years) using only simple patterns and some standard English-language syntactic transformations. However, if you find that you are creating a number of similar patterns to deal with a general problem that could be solved with a transformational rule, you should consider creating such a rule. General rules using the standard KEDS classes such as <**verb**> or <**agent**> can anticipate situations that are not incorporated into the fixed lists of verb phrases. Like verbs and actors, classes and rules can be edited while the program is running, the new constructions tested, and then saved.

8.2 Defining classes and rules

Classes and rules are read from a file specified in the **CLASSES:** command in the *.Options* file. The format of the command is:

CLASSES: '*file_name*'

Example:

CLASSES: 'MidEast89.classes'

The file name should not contain a single-quote but otherwise can be any legal Macintosh file name.

The structure of the class file is:

CLASSES
<***class_name***> **=** ***word***$_1$ ***word***$_2$ ***word***$_3$ **....** ***word***$_n$
<***class_name***> **=** ***word***$_1$ ***word***$_2$ ***word***$_3$ **....** ***word***$_n$
...
<***class_name***> **=** ***word***$_1$ ***word***$_2$ ***word***$_3$ **....** ***word***$_n$
RULES
pattern$_t$ **{** -> | => **}** ***pattern***$_s$
pattern$_t$ **{** -> | => **}** ***pattern***$_s$
...

[2]The KEDS complex pattern specification *appears* to be missing the "iteration" (*) operator found in regular expressions such as those in Unix's *grep*. However, iteration is *implicit* in KEDS's patterns because they automatically skip anything between pattern elements not connected with an underscore. The regular expression's *zero-or-any-number-of-occurrences* operator is accomplished with a space in a KEDS pattern.

pattern$_t$ **{ -> | => }** ***pattern***$_s$
˜ FINISH <any information>

where

class_name = an identifier consisting only of the characters A to Z

$word_i$ = any string of non-blank characters where the first two characters are between '@' and '_' in the ASCII sequence. Words are separated by at least one blank.

$pattern_t, pattern_s$ = any pattern

In addition, any line where the first two non-blank characters are "˜ ˜ " will be treated as a comment and ignored.[3]

Example:

CLASSES
<conjprep> = AS_ ◇ BY_ ◇ WHILE_
<sourceverb> = REPORTED_BY ◇ ACCORDING_TO
˜ ˜ <tobe> will be used in the passive voice rule
<tobe> = IS_ ◇ WAS_ ◇ WERE_
RULES
˜ ˜ convert some prepositions to conjunctions
<conjprep>_<actor>=><conj><actor>
˜ ˜ reverse actors in passive voice
<actor1><tobe>_<verb> BY_<actor2> – ><actor2><verb>
<actor1>
˜ FINISH

◇ = blank character

8.3 Classes

A class list is simply the class name inside < ... > followed by a space-delimited list of the words (or phrases consisting of words connected with "_") in the class. A word can belong to more than one class, and a maximum of 243 classes can be defined.[4]

Class names and words can be entered in lower case but will be shifted to upper case before matching. Unlike most KEDS statements, the **CLASS** statement can be indefinitely long (for example, more than 255 characters) though individual words and phrases are limited to 80 characters. A class definition ends with a <Return> (ASCII 13).

Multiple class statements can be used to define a class; their contents will be accumulated in the class definition. In other words

[3] At present, comments are not carried forward when the classes file is rewritten after classes or rules have been changed using the MODIFY menu.

[4] More generally, the total of the standard classes and the defined classes must be less than or equal to 255.

<claa> = item1 item2 item3
<clbb> = item1 item6 item7
<claa> = item4 item5

has the same effect as

<claa> = item1 item2 item3 item4 item5
<clbb> = item1 item6 item7

This facility is useful if you are temporarily modifying a class dictionary or maintaining several dictionaries that have a great deal of overlap.

> **Note:** Because longer phrases are matched before shorter phrases, **stemming** can cause unintended (and generally unpleasant) results if a full word is used in one class and a stem is used in another. For example, if the *.Actors* file contained
>
> `PALEST [PAL]`
>
> and the *.Classes* file contained
>
> <ARAB_ACTORS> = PALESTINIAN SYRIA EYGPT... then an occurence of "Palestinian" or "Palestinians" always would be classified as belonging to the class <ARAB_ACTORS> but **not** to the class <ACTOR>. This occurs because the dictionary matches "Palestinian" to the class entry PALESTINIAN, but having made this match, it never gets to the shorter stem entry PALEST. To avoid this problem, use exactly the same phrases in all of the files.

8.3.1 Standard Classes

These classes are always initialized and can be used in patterns or rules even if no additional classes are defined.

<ACTOR>	**<AGENT>**	**<VERB>**
<PRONOUN>	**<CONJ>**	**<PREP>**
<TITLE>	**<PLURAL>**	**<COMMA>**

<NUMBER> – see discussion below

<CLAUSE> – used to anchor a pattern to the beginning or end of a clause

<TEXT> – used to anchor a pattern to the beginning or end of the text

8.3.2 <NUMBER> Class

Any set of characters beginning with a digit is automatically assigned to this class. Additional words can be added using to the class file, e.g.

<NUMBER> = one two one_hundred...

Words in the <NUMBER> class can also be assigned default numerical values that are used when computing numeric issues (discussed below); these follow the word inside square brackets, e.g.

<NUMBER> = one [1] two [2] one_hundred [100] hundreds [322]

A default value can be up to 12-characters in length; only positive integers are allowed (i.e. no negative numbers, no decimals). The numerical values are optional – if an explicit value is missing, the numerical value of the word is set to zero

8.4 Composite patterns

Composite patterns can contain sets of alternative matches and the negation of a set of matches; they can be used anywhere in KEDS that a pattern is used. Alternatives and negative are done using the following notation:

8.4.1 Set of alternatives

{ ***pattern***$_1$**|** ***pattern***$_2$**|** ... **|** ***pattern***$_n$ **}**

This will match a string containing

pattern$_1$ `OR` ***pattern***$_2$ `OR ... OR` ***pattern***$_n$.

Unless an alternative pattern ends in "_", it is assumed to end in a space – in other words, an indefinite number of words can be skipped before matching the next element in the pattern.

To force the element following a set to be consecutive, put an underscore after the closing bracket:[5]

{WAS|IS|WILL_BE}_HERE

is equivalent to

{WAS_HERE|IS_HERE|WILL_BE_HERE}

8.4.2 Negation

~ {*pattern_set*}

[5] More generally, appending underscores to the elements of the set {WAS_|IS_|WILL_BE_} HEREwill *not* force the HERE to be consecutive because the "}", rather than the underscore of the matched element in the set, is the pattern element immediately prior to HERE.

This will match anything except *pattern_set*: the pattern fails if any of the subpatterns in the pattern set are encountered. Note that the ~ **must** be followed by a set enclosed by **{...}**. That set can contain strings, classes or patterns. When a pattern fails to match, the search continues from the point where the previous match ended.

A negative set should not end in " **}_** " because an element cannot be consecutive to a failed match. If this is encountered, it will be ignored.

Examples

- SAID WOULD * {<ACTOR> | <AGENT> | <PRONOUN>} - *_TO_ATTACK ~ {UNITED_NATIONS|NATO_FORCES|UNPROFOR}

> **Note:** In order to create an "everything in class <C> except {X}" condition, use
>
> **~ {X} <C>**
>
> This pattern will first try to match {X}, and if {X} is present, it will fail. If {X} is not present, the system will then try to match <C>. For example
>
> **SAID**
> **- ~ {<MEDIASOURCE>} $ * [024]**
>
> **ATTACKED**
> **- ~ {UNO|NATO} * <ACTOR> [223]**
> **- {UNO|NATO} * <ACTOR> [224]**
>
> In the first example, if there is an actor prior to **SAID** that is not in the class <**MEDIASOURCE**>, an event will be generated with a code 024; otherwise no event will be coded from the verb **SAID**. In the second example,
>
> `NATO ATTACKED BOSNIAN SERB POSITIONS`
>
> would be coded 224, while
>
> `CROATIAN FORCES ATTACKED BOSNIAN SERB POSITIONS`
>
> would be coded 223.

8.4.3 <TEXT> and <CLAUSE>

The <**TEXT**> and <**CLAUSE**> tokens can be used to "anchor" a pattern to the beginning or end of the text or to the beginning or end of the current conjunction-delimited clause. These only work when followed by a underscore at the beginning of a pattern (<**TEXT**>_ , <**CLAUSE**>_) or preceded by an underscore at the end of a pattern (_<**TEXT**>, _<**CLAUSE**>); in any other location they will always cause the pattern to fail.

Examples

<**TEXT**>_**$ ***	Only matches a source if it is the first actor in the text
<**CLAUSE**>_**$ ***	Only matches a source only if it is immediately preceded by a conjunction or is the first actor in the text
SAID <**ACTOR**>_<**TEXT**>	Matches if the actor is the final word (or words, when the actor string contains underscores) in the text

8.5 Rules

A rule searches the source text for a target $pattern_t$. If the pattern is found, replaces all of the text between the first and last words matched by the pattern with the replacement $pattern_s$. Rules are applied after classes have been assigned to the words in the text and after **REPLACE** operations but *before* any of the standard syntactic operations (e.g. agent assignment and pronoun dereferencing) are done. Any literal strings are checked against the dictionary and are assigned the appropriate word type if they are found.

The rules are applied in the order they occur in the *.classes* file. There is no way to change this ordering from inside KEDS, but it can be done with an editor.

The left-hand side of a rule (before the -> or =>) can be a pattern of any complexity. The right-hand-side (following the -> or =>) can have only classes and literal strings: it cannot contain alternative sets or *not* operators. Each side of the rule can contain a maximum of 255 characters.

The patterns used in rules cannot contain the *, $, + and % tokens that are used in verb phrases. Use the <**VERB**> class for * and the <**ACTOR**> class for $ and +. Rules are applied before compound actors are coded, so there is no equivalent to the % token.[6]

Unless a replacement class is indexed [see next section], classes are replaced in the same order that they were found. For example the rule

<**actor**><**verb**><**prep**><**actor**> -> <**actor**><**prep**><**verb**><**actor**>

applied to

```
Yelstin stayed in his plane and did not meet with the Irish prime minister
```

would produce

```
Yelstin in stayed Irish prime minister
```

If there is a class in the replacement pattern that does not correspond to a word in the target pattern, the literal

[6] Unless you write an equivalent using a rule, which is possible in the system.

^ <class_name>

is inserted in the replacement text.[7]

8.5.1 Indexing

Classes in a pattern can be indexed by appending a number between 1 and 31 to the class name. This allows a pattern to keep track of where various words occurred in a match and then re-arrange their order in the replacement pattern. For example, applying the rule for handling passive voice in English

<actor1><tobe><verb> BY <actor2> -><actor2><verb><actor1>

to the sentence

ISRAEL WAS ACCUSED BY SYRIA

will produce the assignments

<actor1> = ISRAEL <actor2> = SYRIA

and the result

SYRIA ACCUSED ISRAEL

Because substitution is anchored at the beginning and end of the match, any words that were skipped over in the match will be dropped in the replacement process. For example, applying the passive voice rule to

YESTERDAY THE ISRAELI PRIME MINISTER WAS ACCUSED BY SYRIAN RADIO

will give

YESTERDAY THE SYRIAN ACCUSED ISRAELI RADIO

This substitution has changed the meaning of the sentencealbeit while preserving correct English syntaxbut the source and target would code correctly.[8]

If an indexed class appears in the substitution pattern but not assignment has been made in the target pattern, it is skipped. This usually occurs when the target pattern has a set of alternatives.

[7] Except for the <comma> class, where " , " is inserted.

[8] Note that in this example, correct syntax is maintained only because ISRAELI and SYRIAN can be used as either adjectives or nouns. As with most ambiguous word classes, this can be a liability rather than an asset: if PRIME MINISTER and RADIO were agents the rule would result in an incorrect agent assignment as well. The more complete rule

{<actor1><agent1> | <actor1>}<tobe><verb> BY {<actor2><agent2> | <actor2>}
− >
<actor2><agent2><verb><actor1><agent1>

would solve that problem.

8.5.2 Multiple applications of a rule

Rule may be applied in either of three modes, which are governed by the replacement operator:

- $->$ the rule is applied only *once* to the text.
- => the rule is applied *multiple* times in a sentence until it fails. Each new application begins *after the end* of the previous application. This allows a rule to be applied in multiple clauses but not to modify text that it previously modified.
- >> the rule is applied *multiple times* until it fails; each time the rule is re-applied it starts from the *beginning* of the sentence, so a rule can transform a part of a sentence that it previously transformed.[9]

The ">>" operator can result in an infinite loop: for example

<PREP><ACTOR>>><COMMA><PREP><ACTOR>

would convert

`in Syria`

to

`,,,,,, <infinite times> ,, in Syria`

Similarly,

<actor1><verb><actor2>>><actor2><verb><actor1>

would continue swapping the positions of the two actors indefinitely.

In well-tested dictionaries, infinite loops are usually due to unanticipated constructions in the text that cause otherwise innocuous rules to go awry. To prevent such occurences from causing the program to stop coding, replacement is stopped after the total number of times the rules are applied is equal to twice the number of words in the original text. The program will beep when this occurs. An infinite replacement loop is always obvious if you are looking at the parsed display; if you are autocoding and there are unexplained beeps, this may be the explanation. An infinite replacement may cause other legitimate rules not to be applied, so infinite replacement rules should be avoided as much as possible.

8.5.3 Rule Loops

Rules can be clustered into loops of repeated rules by using **~ REPEAT** and **~ END** commands. A loop is repeated until all of the rules within the loop fail to match, or until the total number of times the rules are applied is equal to twice the number of words in the text. Loops can be nested up to 8 levels deep. For example, in the loop structure

rule$_1$
rule$_2$
˜ REPEAT
    rule$_3$
    rule$_4$
    ˜ REPEAT
        rule$_5$
        rule$_6$
    ˜ END
    rule$_7$
˜ END
rule$_8$
rule$_9$

the outer loop would apply $rule_3$ and $rule_4$, then repeat $rule_5$ and $rule_6$ until both had failed, then apply $rule_7$, then go back up to $rule_3$. When all of the rules in the outer loop had failed, $rule_8$ would be applied. The rules

GHI -> RST
˜ REPEAT
    DEF -> UVW
    ˜ REPEAT
        ABC -> XYZ
        XYZ -> 1XYZ
    ˜ END 1XYZ -> 2XYZ
˜ END
JKL -> PQR

convert

```
DEF ABC DEF ABC GHI JKL MNO GHI JKL
```

to

```
UVW 2XYZ UVW 2XYZ RST PQR MNO GHI JKL
```

The indenting of **˜ REPEAT** loops is optional but improves readability; when rule loops are saved they are automatically indented.

8.6 Notes on Program Speed

Classes are actually more efficient to match than are the text strings found in verb phrases and issues, since all members of a class are stored in the program's dictionary and the assignment is noted when a word is classified. A very large number of classes might slightly slow the dictionary search, but generally this process is quite efficient.

Composite patterns are generally as efficient as simple patterns – the patterns are matched left-to-right with no backtracking and the time required to match a pattern is roughly proportional to its length. Matching will be slightly more efficient if the parts

of the pattern that are most likely to be matched are placed first in an alternative set: {SAID | AFFIRMED | ELUCIDATED} will probably be processed somewhat more quickly than ELUCIDATED | AFFIRMED | SAID , at least if one is coding Reuters.

Rules involve pattern matching on all of the text and therefore will have some effect on the speed of the program if you use a large number. To determine whether this is a significant factor, auto-code a large set of data with and without the rules and compare the speeds reported in the autocoding dialog box after the coding is stopped.[10]

8.7 Examples

8.7.1 Passive Voice

The granddaddy of all rules is the English-language passive-voice transformation. The safest version of this looks like:

<TOBE> = HAVE_BEEN_ IS_ WAS_ WHEN_ WILL_

<ACTOR1>_<TOBE>_<VERB> BY_<ACTOR2> – >
<ACTOR2> <VERB><ACTOR1>

The one potential problem with this formulation is that the actor must come immediately before the <TOBE> verb, which will miss cases where the actor identification is actually an adjective (e.g. ISRAELI PRIME MINISTER). By eliminating the underscore after <ACTOR1>, the rule can be made more general, but at the risk of making incorrect applications. For example the rule with **<ACTOR1> <TOBE>**... applied to

```
970130 REUT-0024-01
Six European countries formally agreed on Thursday to increase the number of
observers in the West Bank town of Hebron , most of which was handed over to
Palestinian rule by Israel two weeks ago.
```

reduces the sentence to

```
Six Israel agreed European two weeks ago.
```

8.7.2 Using rules and classes in attribution statements

One of the most common problems in news reports are attributed statements

```
Kuwait newspapers reported that the Saudi foreign minister plans to visit the
United States...
```

[10] When autocoding, almost half of the time is taken up by reading and writing data, and basic operations such as assigning classes to the individual words in the text, so changes in the coding process itself often have relatively small effects on the speed of the coding.

If one wishes to code attributed statements as events – which would be consistent with the fact the statements made by Reuters are being coded as events – then sentences such as this cause a problem: The source of the *statement* (Kuwait) rather than the source of the *event* (Saudi Arabia) is the first actor in the sentence, and therefore coded as the source, so even if REPORTED has a subordinate code, one gets the incorrect event:

Conveniently, Reuters usually puts attribution at the end of a sentence, where this problem does not occur. This is not true of all newswire sources, however. To deal with attribution at the beginning of a sentence, one could use the following classes and rules:

<attrverb>= SAID REPORT CLAIM

<newsource>= NEWS RADIO TELEVISION

<actor1> <source> <attrverb> <actor2> <verb> <actor3> ->
<actor2> <verb> <actor3>

Under these rules, the source text given above would be converted to

```
Saudi plans to visit United States...
```

and coded

SAU <plans to visit> USA

8.7.3 Compound Verbs

Reuters occasionally uses compound verbs such as "intercepted and detained" in the example below:

```
970108 REUT-0008-01
A Croatian naval patrol intercepted and detained an Italian fishing boat in
the northern Adriatic for illegal fishing in its territorial waters, the state
news agency HINA said on Wednesday
```

These cause a problem when coding by clause because the target search will not cross the conjunction AND_. The following rule will shift the target back in front of the conjunction:

<VERB1>_<CONJ>_<VERB2> <ACTOR1> ->
<VERB1> <ACTOR1> <CONJ> <VERB2> <ACTOR1>

This converts the text about to

```
A Croatian naval patrol intercepted Italian and detained an Italian fishing
boat...
```

Chapter 9

FORMATTING OUTPUT

9.1 Purpose

KEDS provides a great deal of flexibility of formatting of the output on the screen and in the output text file; this is particularly useful if you intend to use the output in a database, spreadsheet or statistics program. The default formats used by the program are sufficient for most simple event data work, but if you are going to be coding something more elaborate, you will probably want to customize the formats.

9.2 Formatting Commands

The format of the information written to the *.events* file and displayed in the event window are controlled by the *.Options* commands

OUTPUT:*format string*

DISPLAY:*format string*

The *format string* is simply a list of the output variable names in the desired order, separated with additional characters as needed. The format begins immediately after the ":".

Variable names:

DATE	Date
SOURCE	Source
SOURAG	Source agent
TARGET	Target
TARGAG	Target agent
EVENT	Event code
CODEXT	Event code text (set using a CODE: command)
PHRASE	Phrase matched by event
NOTES	Notes field (see below)
ISSUEk	Issue (e.g. ISSUE3, ISSUE9, ISSUE0)
CODER	Coder ID and date
IDINFO	Record ID

OUTPUT only:

TEXT: Text (see below)

The variable names in these commands are case-sensitive **SOURCE** will be recognized as a variable name but **Source** will be assumed to be a string separating variable names. A maximum of 32 format elements can be specified, and the total length of the format string must be less than 255 characters.

Blank characters are significant in formats! A blank in the format, even one separating the variable names, will be interpreted as producing a blank in the output. Note also that any misspellings of the variable names will be interpreted as strings between variables. If you need to use the actual words EVENT etc., just use something else (e.g. EVXNT) and do a global replace in an word processor. Lower case will also work.

> **Note:** If you incorrectly use the **TEXT** variable in the **DISPLAY** format, you will see the string **#%TEXT%#** rather than the actual text. If for some reason the string **#%TEXT%#** is used in your format, the text will be placed at the first occurrence of it (in other words, don't use **#%TEXT%#** in a format)

9.2.1 Specifying the length of variables

If **[n]** is appended to the variable name – for example

SOURCE[6] EVENT[3] TARGET[6] CODEXT[20]

then the length of the variable is fixed at n characters. Unless the variable is a numerical ISSUE, when the number of characters entered is longer than the length, the final characters will be eliminated; if shorter, blanks will be added to the end of the string to get the correct length. If no length is specified, the maximum code length of 12 characters will be used except for **EVTEXT, PHRASE, NOTES** and **TEXT**, which are displayed at their full length.

Numerical issues (those with an **OPTIONS='NUMERIC'** command) are *right-justified* blanks or zeroes are added to the beginning of the number until it is the specified length Blank-filling is the default; zero-filling is done if the **OPTIONS='NUMERIC ZERO'** command was used. If the number is longer than the specified length, the left-most digits are removed and the first remaining digit is replaced with a * to indicate that it was truncated.

> **Note:** This rule does not apply to the NOTES field; see the description below.

Example

SOURCE = **ISRMIL**
TARGET = **PALDEMO**
EVENT = **221**
Format: **SOURCE[8]+PALDEMO[6]==>EVENT[2]**
Output: **ISRMIL◇◇<tab>PALDEM==>22**

9.3 Default formats

If no formats are specified, the following defaults are used:

DISPLAY:SOURCE[6]◇◇TARGET[6]◇◇EVENT[3]◇◇(CODEXT[10])◇◇PHRASE[30]
OUTPUT:DATE◇◇SOURCE[6]◇◇TARGET[6]◇◇EVENT[3]◇◇CODEXT

where ◇=blank. Note that the default formats show only 6-character codes for source and target and 3-character codes for the event, rather than the 12-character maximum.

9.4 Inserting control characters in formats

When a format string is interpreted, the characters ^ , +,/ and ! have a special meaning:

+	insert tab (ASCII 9)
/	insert carriage return (ASCII 13)
!	insert line feed (ASCII 10)
^ +	insert '+'
^ #	insert '#'
^ /	insert /
^ !	insert !
^ ^	continuation (character combination occurs at the end of the previous line and the beginning of the next line)

The continuation is used to link parts of a format string that occur on two consecutive lines. The two-line construction

OUTPUT: This is the first part of the format,^ ^
^ ^ this is the second

is equivalent to

OUTPUT: This is the first part of the format, this is the second

In a long format string, this facility allows you to break the command at points that can be easily read rather than depending on the formatting of the text editor. Continuation also allows the command describing the format to be longer than 255 characters (due to the variable names) even though the format itself can be a maximum of 255 characters.

Examples of the use of the other special characters are given below.

9.5 NOTES

The NOTES field is designed to be used in machine-assisted coding to contain information that can only be ascertained by a human coder. When the **SET: NOTES=TRUE** command is used in the *.Options* file, a Notes box is shown in the event editing window. The program does not put any information in the box; the content must be provided by the coder.

The display and output of the NOTES string is controlled by a special format string where

#	insert next character from the string entered by the coder
^ nnn^	equivalent to nnn # symbols; nnn can be one to three digits
^ , ^ ◇, or ^ /	if this occurs at the beginning of the string, remove commas, blanks or slashes before processing string.

This allows a coder to place delimiters between the fields in the NOTES string.

anything else insert that character

If there are more # characters in the format string than there are characters in the string entered by the coder, blanks replace the # characters in the output.

The total length of the NOTES entry and the NOTES format string must be less than 254 characters.[1]

Example

Notes string: 1234567890

Format string: **NOTES[##+###+ X##+#,###]**

Output string: **12<tab>34<tab>** ◇◇ **X56<tab>7,890**

Example

Notes string: 12/34/56/7890

Format string: **NOTES[^ /##+###+$^ 8^ +#,###]**

Output string: **12<tab>34<tab>$56**◇◇◇◇◇◇ **<tab>7,890**

9.6 TEXT

The **TEXT** field shows the text that was coded. Because the text is potentially very long, and it is already displayed in the text window, this option can only be used in an **OUTPUT** format string. The **TEXT** field can have the following formats

[1]The NOTES text entry field is set to accomodate about 16 characters. If you need it substantially larger and know a Mac junkie who can use Apple's *ResEdit* program, the following patch will work:

1. Open DLOG/DITL resources #439 ("Edit Event")
2. Vertically enlarge the DLOG box
3. In the DITL editor, select all of the "Issue" edit boxes (item numbers 12 through 21) and move these down as a block, keeping their relative positions fixed.
4. Enlarge the Notes text edit box (item #11) and put it wherever you would like.

The event editing dialog should accomodate itself to these changes, repositioning the dialog box and the Notes and Issues labels.

TEXT	Write the text as a single string with no delimiters
TEXT[nn]	Write only the first nn characters of the text
TEXT[nn:*characters*]	Write the text in lines less than or equal to nn characters in length, broken on words, with the character string *characters* at the end of each line except the last. The special characters (**+**,**/** and **!**) can be used; this provides some flexibility in formatting the source text to be compatible with other software.

nn must be less than 255.

Examples:

Source text:

```
Iraq has concentrated nearly 100,000 troops close to the Kuwaiti border, more
than triple the number reported a week ago.
```

OUTPUT:TEXT[30]...+

Iraq has concentrated nearly 1...<tab>

OUTPUT:TEXT[50:!>>>]

Iraq has concentrated nearly 100,000 troops close <lf>
>>>to the Kuwaiti border, more than triple the number <lf>
>>>reported a week ago

9.7 Examples of complex formats

In the examples below, <lf>=line feed; <tab>=horizontal tab, ◇=blank. Due to the formatting of this document, there are some physical line feeds in the examples where these would not occur as <lf>s in the text written to a file.

Original source text:

```
891209ID0006◇◇◇◇◇Reuters
Jordanian◇riot◇police◇using◇tear◇gas◇battled◇Palestinian◇demonstrators◇
in◇a◇refugee◇camp◇near◇Amman◇on◇Saturday◇for◇the◇second◇night◇running,◇
witnesses◇said.
```

OUTPUT:DATE (IDINFO)/TEXT[40:/]/SOURCE SOURAG[3] TARGET TARGAG[3]^ ^ EVENT[3] ISSUE1[3] ISSUE2[1]//

891209◇◇(ID0006)
Jordanian◇riot◇police◇using◇tear◇gas◇ <lf>

battled◇Palestinian◇demonstrators◇in◇a◇ <lf>
refugee◇camp◇near◇Amman◇on◇Saturday◇for◇ <lf>
the◇second◇night◇running◇,◇witnesses◇ <lf>
said◇ <lf>
JOR◇◇◇◇◇◇PLC◇◇◇◇PAL◇◇◇◇◇◇◇DEM◇◇◇◇222◇◇◇◇REF◇◇◇◇8

OUTPUT:DATE+SOURCE+TARGET+EVENT[3]+CODEXT+TEXT[30]'

891209<tab>JOR◇◇◇ <tab>PAL◇◇◇ <tab>222<tab>NONMIL◇DES
TR<tab> ◇Jordanian◇riot◇police◇using◇te

OUTPUT:DATE:+SOURCE[3](SOURAG[3])◇ ==> ◇EVENT[3]◇ ==> ◇TARGET[3](TAR
GAG[3])◇+Phase:PHRASE[30]

891209:<tab>JOR(PLC)◇ ==> ◇222◇ ==> ◇PAL(DEM)<tab>Phas
e:BATTLED◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

OUTPUT:DATE+SOURCE+EVENT+TARGET+NOTES[##+#+###+###]

891209<tab>JOR◇◇◇ <tab>222◇◇◇ <tab>PAL◇◇◇ <tab> ◇AB<tab>1
<tab>CD<tab>345

9.8 LABELS

The **LABELS** command can be used to set the heading labels that appear in blue in the event window. The format of the command is

LABELS: $label_1$+$label_2$+...+$label_n$

where $label_1, label_2, ..., label_n$ are the text labels for the variables, listed in the order they occur in the **DISPLAY** command and separated by "+." Except for the labels, the header has the same format (including intervening text and the number of characters displayed) as the **DISPLAY**.

If the **LABELS** command is not used, the heading defaults to the variable names. If fewer labels are listed than variables are displayed, the remaining labels are set to the defaults; if more labels are listed than variables are displayed, the remainder are ignored.

Example

The commands

DISPLAY:SOURCE (SOURAG)◇ ==> ◇TARGET◇(TARGAG)◇:◇:◇EVENT[3]
LABELS: Source+Agent+Target+Agent+Evt

produces the heading label

Source(Agent)◇ ==> ◇Target◇(Agent)◇:◇:◇Evt

9.9 Text File Output

If the command **SET: TEXT FILE=TRUE** is included in the *.Options* file, a separate file will be produced that contains

DATE<tab>IDINFO<tab>TEXT

This file is identified with a *.text* suffix. The text record will be written to this file once for each case where at least one event has been coded from the text. This file is designed to be used with a relational database program to associate events with the coded text and is an alternative to including the text with every event record (which creates very large and redundant files).

9.10 Problems File Output

If the command **SET: PROBLEMS FILE=TRUE** is included in the *.Options* file, a separate file will be created to save for later analysis problematic records identified by the coder. This file is identified with a *.prob* suffix; the files are numbered according to the coding session and are only retained when they contain the record of at least one event. *.probs* records contain the original text, the parsed version of the text, the coded events and any additional comments the coder might wish to add (see the Modify/Comments menu option).

9.11 PANDA Formatting

If the command **SET: PANDA FORMAT=TRUE** is included in the *.Options* file, the formatting defaults to an assortment of formatting conventions used by the PANDA project. These are:

- The missing value code *** and the null code - - - are replaced with blanks in the output file;
- At least one blank record is written for each source text, even if no events are found;
- Actors with codes beginning with '7,' '8,' or '9' cannot be assigned to the PLACE variable [PANDA uses these for international organizations and other non-territorial actors];

If not otherwise set by other *.Options* commands:

PANDA **OUTPUT** format:

DATE+IDINFO+SOURCE[6]+SOURAG[3]+TARGET[6]+TARGAG[3]+ EVENT[3]+ISSUE1[3]+ISSUE2[1]+ISSUE3[1]+ISSUE4[1]+ISSUE5[1]+ ISSUE6[1]

PANDA **DISPLAY** format:
SOURCE[**6**]◇**SOURAG**[**3**]◇◇◇**TARGET**[**6**]◇
TARGAG[**3**]◇◇◇**EVENT**[**3**]◇◇◇**ISSUE1**[**3**]◇**ISSUE2**[**1**]◇**ISSUE3**[**1**]◇**ISSUE4**[**1**]◇
ISSUE5[**1**]◇**ISSUE6**[**1**]◇◇**CODEXT**

PANDA **ISSUE Titles:**

Issue1 Title = 'Place';

Issue2 Title = 'Issue';

Issue3 Title = 'Domain';

Issue4 Title = 'Contxt'

Chapter 10

AGENTS

10.1 Purpose

In some circumstances, it is useful to differentiate the agent responsible for an event, for example distinguishing "Israeli soldiers," "Israeli police," and "Israeli settlers." "Agents" are typically improper nouns with associated codes:

AGENT: DISSIDENT [OPP]
AGENT: ELECTORATE [CON]
AGENT: EMIGRANT [REF]
AGENT: EMIGRES [REF]
AGENT: ENVIRONMENTALIST [ENV]
AGENT: ENVOY [EMB]
AGENT: FARMER [LAB]

KEDS can assign an agent to both the source and target; these fields are defined using the **OUTPUT** and **DISPLAY** commands. This facility is activated by setting **SET: CODE AGENTS=TRUE** in the *.Options* file.[1]

10.2 Defining Agents

Agents are defined using the AGENT command in the *.Options* file

AGENT: ***string [code]***

See example above.

[1] KEDS versions 0.3 and 0.5 contained a very different **AGENT** facility designed to compensate for implicit actor identification based on locations and interactions (e.g. `POLICE IN GAZA ARRESTED DEMONSTRATORS` would use the location, Gaza, to identify "police" as Israeli and "demonstrators" as Palestinian). This did not work particularly well and was largely unnecessary in Reuters, so the facility was discontinued after version 0.5.

10.2.1 Implicit Agents

An implicit agent can be associated with the actor. This is typically done with proper names. The implicit agent causes an agent to be assigned to the actor even WHEN no agent occurs explicitly in the sentence. The implicit agent code follows the actor code, separated by a colon:

GEORGE BUSH [USA:PRES]
BOUTROS GHALI [UNO:SG]

Implicit agents can also be used in coded compounds:

PRESIDENT_AND_VICE-PRESIDENT [USA:PRES/USA:VPRS]

10.3 How Agents are Assigned

The program attempts to associate an actor to all agents found in the text, using the following priority:

1. Implicit agents (e.g. GEORGE BUSH [USA:GOV])
2. <actor><agent> (e.g. `FRENCH POLICE`)
3. <agent><preposition><actor> (e.g. `POLICE IN RANGOON`)
4. an actor within ± 2 words of agent; the number of word can be adjusted using the **SET: AGENT SEARCH** command

If none of the above conditions are found, the agent is assumed to an actor itself, and is treated as such when identifying the source and target. In a statement like

```
Police fought demonstrators last night outside of the parliament, official
Polish sources reported
```

will produce the coding

*** POL *** DEM.

In this and most other instances the national identity of the agents can be determined by information coded elsewhere in the sentence using a **PLACE** issue.

An agent that has been converted to an actor is called an implicit actor and is denoted by italics in the parsed display of the text. The rule of not matching identical source and target applies to [actor:agent] combinations, not just to actors alone.

> **Note:** Implicit actors are detected in the parsing of syntactic compound actors just as regular actors are, so
> `POLICE AND DEMONSTRATORS CLASHED`
> will form the compound POLICE_&_DEMONSTRATORS. KEDS does not deal with the ambiguous form of compound agents is found in phrases of the form:

```
AMERICAN AND ISRAELI NEGOTIATORS
(= AMERICAN NEGOTIATORS AND ISRAELI NEGOTIATORS)
```

Assuming AMERICA is an actor, for KEDS this sentence is syntactically equivalent to

```
JAMES_BAKER AND ISRAELI NEGOTIATORS
```

and the extension of agent assignment would not be appropriate in the second situation. If the text you are coding warrants the use of compound agent assignment, use the rule

<actor$_1$ >_AND <actor$_2$ ><agent>=>
<actor$_1$ ><agent> AND <actor$_2$ ><agent>

10.4 Displaying and Writing Agent Codes

Unless the **SET: PANDA FORMAT=TRUE** command has been used, the default **DISPLAY** and **OUTPUT** formats do not include the agent codes. To change this, include **DISPLAY** and **OUTPUT** commands in the *.Options* file.

10.5 Modifying the Processing of Agents

The treatment of agents is controlled by the following parameters that can be modified using the **SET** command in the *.Options* file.

10.5.1 CONVERT AGENTS [default: TRUE]

TRUE: If an agent cannot be assigned to an actor, it is converted to an implicit actor with a null identity.

FALSE: Unassigned agents are not converted to actors.

In Reuters text, this setting has a substantial effect on the assignment of sources and targets.

10.5.2 REPLACE IMPLICIT AGENTS [default: FALSE]

TRUE: If an actor with an implicit actor code can be assigned to an agent according to the assignment rules, that agent overrides the implicit agent assignment

FALSE: The implicit actor assignment is retained, but the agent is treated as if it had been assigned an actor (in other words, it is not later converted to an implicit actor)

Example

Actor: **CLINTON [USA:GOV]**
Agent: **AGENT: AMBASSADOR [AMB]**
Source Text:
`President Clinton's ambassador to the GATT talks...`

REPLACE IMPLICIT AGENTS = FALSE−− >source is USA GOV

REPLACE IMPLICIT AGENTS = TRUE−− >source is USA AMB

10.5.3 AGENT SEARCH = (<back>, <frwd>) [Default: (2,2)]

Controls the number of words to look before and after an agent to find the actor.

Chapter 11

ISSUES

11.1 Purpose

KEDS can code up to 10 "issues" based on simple (rather than syntactic) pattern matching. Issues are typically strings used to code aspects of the context of an event, for example

ABORTION [T]
AIDS_ [E]
ANCESTRAL LAND [N?]
APARTHEID [H!]
ASYLUM [H]
BALLOT [G?]
BAN_THE_DEATH_PENALTY [P?]
BANKING [F!]

though an *.Issues* file can contain strings dealing with any topic. Issue coding with long lists is v e r y s l o w because all of the strings must be checked. In contrast, coding issues by linking them to verb phrases [see below] is almost as fast as simple event coding. The PANDA coding scheme makes extensive use of issue codes and their project has developed several dictionaries containing issues lists dealing with domestic political activity.

11.2 ISSUE Command

The ISSUE commands in the *.Options* file are ISSUE1, ISSUE2, ... ISSUE9, ISSUE0 with the syntax:

ISSUEn: TITLE='string' FILE='string' DEFAULT='string'
OPTIONS='string' TAG='string'

The *strings* in the command are between single quotes; the quote must be the first non-blank character after the equal sign.

TITLE gives the name of the issue (up to six characters) that will appear in the Edit dialog. If **TITLE** is not specified, **ISSUEn** is used as the title.

FILE is the name of a TEXT file in the same folder as the other files. Reading strings from a file is optional; you may assign an issue by only using a default.

DEFAULT gives the default code; this must be ≤ 6 characters in length.

If the **DEFAULT** string begins with **@** followed by one of the following variable names:

DATE	**SOURCE**	**SOURAG**,
TARGET	**TARGAG**	**EVENT**

(for example **@EVENT** or **@SOURCE**), then the default value for the issue is set to the value of that variable. For example, in **ISSUE1** in the example below, the default value assigned to "Place" is the code for the source.

OPTION='*string of options*'

NONRES: search for the string in the comma-delimited nonrestrictive clauses as well as the rest of the text.
Default: don't look at these clauses

NUMERIC: total the numbers that occur immediately prior to the string specified in the TAG field; this is described in detail below

ZERO: Right-justify numeric issues using zeros.
Default: right-justify numeric issues using blanks.

TAG='*numeric tag string*'

This is used to identify the numbers that should be coded in a numeric issue; it is described in more detail below.

Example
ISSUE1: TITLE='Place' FILE= 'PANDA.PLACE' DEFAULT='@SOURCE'
ISSUE2: TITLE='Issue' FILE= 'PANDA.ISSUES' DEFAULT='X'
ISSUE3: TITLE='Domain' FILE= 'PANDA.DOMAIN' DEFAULT='9'
ISSUE4: TITLE='Contxt' FILE= 'PANDA.CONTEXT' DEFAULT='0'
ISSUE5: TITLE='Partic' FILE= 'PANDA.NUMBERS' DEFAULT='0'
OPTIONS='NUMERIC ZERO' TAG='#D#'

11.3 Issue Priorities

The issues file contain lists of strings and their associated codes. Issues files cannot contain composite patterns or actor tokens – string matching is affected only the underscore (words must be consecutive) and blank (allow intervening words).

> **Note:** If you want to code an issue using a composite pattern, this can usually be done by using a rule to insert a special string into the text that signals the presence of that pattern, then coding that string.

Issue codes can be designated as dominant or subordinate using the **!** and **?** operators. The code for the issue with the highest priority will be assigned to the event. In addition, a numerical priority between **-254** and **255** can be assigned to a keyword by putting the value in **nnn** after the code

ABORTION [**T<10>**]
AIDS_ [**E<123>**]
ANCESTRAL LAND [**N?**]
APARTHEID [**H!**]
ASYLUM [**H<-54>**]

The standard dominant code (**!**) is assigned a priority value of 255; the standard subordinate code is assigned a priority value of -1. The issues are searched in the order that they are entered in the file, and if two issues are found that have the same priority value, the first one encountered will be used.

As with actors and verbs, null coding can be used to eliminate problematic phrases. For example

MOVIE BOMBED [**- - -**]
BOMBED [**MILACT**]

would coded all phrases containing BOMBED as [MILACT] except for the phrase `MOVIE BOMBED`. The null-coded phrase should come before the regular phrase.

11.4 NUMERIC Issues

Numeric issues are designed to total numbers that occur in a sentence; it can be used to get the size of a protest, the casualties in a military engagement and so forth.

A numeric issue looks for a word of class <NUMBER> followed immediately by the string specified in the TAG='string' field. The numeric issue is set to the total of the numbers that are followed by the tag.

In some instances, the tag can be a natural language word, e.g. **TAG='DEMONSTRATORS.'** More typically – in order to deal with synonyms – it will be an artificial tag set by a class and rule:

<demos> = DEMONSTRATORS PROTESTERS STUDENTS

<number>_<demos>=><number> #D# <demos>

This combination of a class and rule would place the tag string **#D#** (which is unlikely to occur naturally in the text) immediately after any number that was followed by the words "DEMONSTRATORS," "PROTESTERS," or "STUDENTS." An ISSUE command could then use TAG = '#D#' to total those numbers.

Numbers are interpreted according to the following rules:

1. The total must be less than $10^{13} - 1$; in other words, it must fit in a 12-character code string.
2. Commas inside numbers are ignored (e.g. 10,000 is read as 10000)
3. Negatives and anything after a decimal place are ignored: numeric issues only deal with positive integers;[1]

The file specified in the **FILE=*'string'*** statement contains the numerical equivalents of quantitative vocabulary, e.g.

THOUSANDS [3162]
HUNDREDS [316]
MANY [500]
LARGE_CROWDS_OF [1000]

These equivalents can differ between issues: "many police" does not have to be the same number as "many demonstrators." If a numerical word is found that does not have an entry in the FILE, then the default value set in the *.Class* file (if any) is used. All of the words in the file are entered into the dictionary as having the <NUMBERS> class and will be included in the <NUMBER> class list if a class file is written.

The DEFAULT sub-command has no effect in a numeric issue: words in the FILE list must be assigned numerical values, and numbers not on that list are either interpreted literally ('123') or according to the default value assigned in a *.Class* file.

11.5 PLACE:

PLACE is a special issue developed for the PANDA project that is activated with the **SET: CODE PLACE = TRUE** command in the *.Options* file. When **PLACE** is activated, the following rules are used to identify the *location* where the event occurred

1. Look for a preposition (these are set using the <**prep**> class in a *.Classes* file or a **PREP** statement in the *.Options* file) followed within two words by an actor whose code does not begin with:

   blank PANDA's missing value code

   - null code

   7, 8 or 9 PANDA's prefixes indicating individual, organizational and multi-lateral actors

[1] If someone really needs negatives and decimals, the program could be changed; but it is faster this way and the intention of this facility is to count people, who generally occur in quantities codeable as positive integers. With the clever use of rules and replace statement and some post-filtering in a database or spreadsheet, it should be possible to get around the restriction on integers, for example if you were dealing with monetary amounts.

If such an actor exists, assign that actor as the **PLACE**

Example
`Israeli Prime Minister Rabin and Jordan's King Hussein met` *`in Washington`* `today.`

Note: Prepositional phrases often occur in indirect objects, for example

`Clinton and Yeltsin discussed the war in Bosnia`

so this rule produces quite a few incorrect identifications.[2]

In addition to this problem, a number of short English prepositions – for example TO_, OF_, BY_, AT_ – can function as placeholders, tense markers or as parts of idiomatic expressions. For example, BY is a preposition, but also a marker for passive voice, and the *Random House College Dictionary* (1975) lists 27 additional meanings for BY. There are 31 distinct meanings for IN and 25 meanings for TO. Treating IN and TO as if they occurred *only* as prepositions is a less than completely reliable strategy. A fluent speaker disambiguates, without conscious processing, these uses by context:

`In a week, the negotiators appointed by the UN are going to the meeting to be held in the village by the airport`

A computer does not have this capability.

2. Look through all of the coded events for a source whose code does not begin with characters mentioned above.
3. Look through all of the coded events for a target whose code does not begin with characters mentioned above.
4. Look for a phrase in the first issue list. In the PANDA setup, this is designated as the **Place** list

   ISSUE1: TITLE='Place' FILE='PANDA.PLACE'

   On the assumption is that anything in the ISSUE1 list can be a legitimate place, the codes are not checked for a legal first character. PANDA reserves the

[2] This example, of course, cannot be unambiguously parsed because one syntactically legal interpretation would be

`In Bosnia, Clinton and Yeltsin discussed the war`

If that were the interpretation (highly unlikely in 1994; perhaps possible in 1996), KEDS's **PLACE** rules would be correct; the sentence is syntactically equivalent to

`Clinton and Yeltsin discussed the Bosnian war at the UN.`

Marx provides better-known example of this ambiguity:

`Last night I shot an elephant in my pajamas. How he got into my pajamas I'll never know.` (Groucho Marx, *Animal Crackers*)

ISSUE1 list for purely geographical terms that cannot be identified with a single actor, e.g. "Middle East," "Persian Gulf," "Jordan River".

The **PLACE** rules are evaluated in the order given and the first rule that succeeds establishes the location. If none of these hold for any of the events, the **PLACE** is set to missing.

The first **PLACE** that is successfully identified by these rules is assigned to all of the events coded from a lead. In other words, if a lead generates events with multiple sources (for example, through a compound source), the first source identified will be assigned as the **PLACE** in all of the leads.

11.6 Verb-linked Issues

In some situations, issues can be coded directly from a verb phrase. To accommodate this, KEDS allows for a verb to have multiple codes, using a format similar to that used for compound actors:

APOLOGIZE [013/H/12/M]

The issue codes are inserted into the *active* issues (in other words, those with an **ISSUEn** statement in the *.Options* file) in order, *starting with* **ISSUE2**. This convention is used because **ISSUE1** is reserved for **Place** in the PANDA coding scheme and therefore is coded by the other routines. The example above would assign:

ISSUE2=H ISSUE3=12 ISSUE4=M

Because the physical order of the issue codes in the output file can be changed using the **DISPLAY** and **OUTPUT** options, this logical order doesn't impose any real restriction on the use of issues.

If you don't want to code an issue for a particular verb phrase, use the null code [- - -], and the issue will be filled in using the issues vocabulary lists. For example in

APOLOGIZE [013/- - -/12/M]

one would get

ISSUE2=<filled using issues lists>
ISSUE3=12 ISSUE4=M

Verb-linked issues also work with paired codes – just separate the two halves of the pair with a colon:

VISIT [032:033/H:I/123/J:K]

When the primary event code is paired, the system checks whether any of the issue codes are paired. If they are, the first code is assigned to the *source-target* event, and the second code is assigned to the *target-source* event. If a code is not paired, it is assigned to both events. To deal with a situation where the issues are paired but the event code is not, just assign the same event code to each half of the pair.

VISIT [031:031/H:I/123/J:K]

When issues are assigned through a verb, these follow the dominant/ subordinate status of the *verb*; additional dominant/subordinate status codes on the issues will be ignored.

Chapter 12

PROFILES

12.1 Purpose

The **PROFILE** command generates a record showing whether a word or phrase appeared anywhere in the source text: this can be used for non-syntactic content analysis. The difference between a profile and an issue or event is that profiles ***count*** the occurrences of a phrase anywhere in the source text, whereas events and issues identify a ***single code***. Profiles are primarily designed for statistical indexing and the determination of the context (see Salton 1989 for a discussion of these methods) and are a useful supplement to the syntactic content analysis that is the primary function of KEDS.

The profile records are written to a file with the suffix *.profile*. Each record consists of the source text identification (date and identification code), followed by a set of codes and, usually, numbers. The codes corresponding to the actors, verb phrases, issues and other strings found in the text. The numbers can be either

1. the *frequency* of the phrases generating the code: for example, if phrases corresponding to the code **ISR** occur four times in the text, the profile will contain

   ISR 4
2. the *location* of the code in the text: one might have

   ISR 1 ... ISR 10 ... ISR 20 ... ISR 25
   assuming the phrases that corresponded to **ISR** occurred in the first, tenth, twentieth and twenty-fifth words of the text.

The profile does not record default values of issues; it only records codes corresponding to phrases actually in the text. Null codes are not included in the profile.

Because the frequency of codes and the ordering of codes varies for each record, the records in the *.profile* file will usually not be directly usable in a statistical program

but will have to be converted to a rectangular format first using a simple computer program.

12.2 PROFILE command

The syntax of the **PROFILE** command is

PROFILE: ISSUEn1 ... ISSUEnm VERBS ACTORS AGENTS

{ **LOCATION** / **CODE ONLY** } **DELIMITER=<char> FIXED=n**

All sub-commands in the **PROFILE** command are optional; the order of the sub-commands does not matter. The abbreviation of the command is in bold face; for example **FIXED** can be abbreviated **FIX**.

ISSUEn: Include codes from ISSUEn in the profile; as many ISSUEs can be included as needed.

VERBS: Include codes from all non-null verbs in the profile. This records only the verb roots, not the phrases.

ACTORS: Include codes from all actors in the profile.

AGENTS: Include codes from all agents in the profile.

The **VERBS, ACTORS** and **AGENTS** codes are written to the profile only if they are simple codes (i.e. these cannot contain compounds, embedded issues or date restrictions).

12.2.1 Formatting:

FIXED=n: All codes are adjusted to be n characters in length; they are truncated if their length is greater than n, and blank-filled if the code length is less than n. The default value of n is 6.

default: Codes are written at their full length, without blank filling.

12.2.2 Numbers

LOCATION:	The number following a code is the location of the word that generated the code as counted by words from the beginning of the text.
CODE ONLY:	No location or frequency numbers are written.
default:	The number following a code is the frequency that code appeared in the text.

All numbers are written in a fixed format with 4-digits, right-justified.

12.2.3 Output Delimiter

DELIMITER=<character>:	Character separating the entries. The formatting characters +, ! and / work as described in the chapter Formatting Output
Default:	+ (tab)

Example:

DELIM=	sets the delimiter to a space
DELIM=/	sets the delimiter to a line feed
DELIM=^ /	sets the delimiter to /

12.2.4 Command Example

PROFILE: VERBS ISSUE2 ISSUE5 ISSUE6 DELIMITER=, FIXED=8

Effect: The *.profile* file will contain all of the codes from verbs and from ISSUES 2, 5 and 6. The number following the code will be the frequency of the code in the source text, the output fields will be separated by commas (,), and the codes will be fixed in length at 8 characters.

> **Note:** A maximum of 256 codes can be recorded in a profile record for a single text. In the unlikely event that more than 256 codes are found, the final code will be $OVERFLOW and the number following this will be the number of codes found beyond the 255th.

Chapter 13

INTERFACE

13.1 Purpose

This section will discuss the operation of the KEDS program, including the windows, mouse and keyboard controls, menus and dialogs. KEDS operates using a standard Macintosh interface and with a few exceptions, follows the standard Macintosh user interface guidelines. This section assumes that you are familiar with basic operations such as pointing, clicking and responding to dialogs.

13.2 Starting the Program

The KEDS program is started by double-clicking either the KEDS program icon or the icon of a KEDS project file. An introductory screen will be displayed, followed by a dialog box asking for the coder ID. Enter any string (e.g. your initials) in response to this, then click OK. The coder ID and date of the coding session are recorded in the project file.

If you double-clicked the program icon, the next dialog box asks whether you want to use an existing coding file or initialize a new file. Clicking the **Use existing project file** button presents a standard Macintosh file selection dialog listing KEDS project files. You can use this dialog to move around among the folders on the disk; the program does not need to be in the same folder as the project file, dictionaries and data. Open the appropriate project.

Clicking the **Initialize new project file** button creates a new project file or reset an existing file. This process is described in the chapter Input Files/Project/Initializing Project Files.

The program will now present information on the current project status and begin reading the *.Verbs, .Actors, .Options* files. If you are using long dictionaries, the

input process is quite slow – typically three or four minutes. If you are using long dictionaries and an older Macintosh (e.g. an SE/30), the process is really, really slow, so read your mail or get a cup of coffee. The delay is due to KEDS setting up a variety of internal data structures as it reads the dictionaries. The system will "beep" when the input process is completed.

After reading the dictionary information, the program may then present a dialog asking whether you want to skip previously-coded record. If you click **Yes**, the program will advance to the input record following the final record coded in the previous coding session (this can also take a while on a slow machine). If you click **No**, coding will begin with the first record in the text file.

When you reach the end of the text file, you will be given the option of coding another file. If you click **OK**, the program will present a text file selection dialog; if you click **Cancel**, the program will go through its standard **Quit** routine, closing the files and saving any changes in the directories.

13.3 Windows

KEDS usually displays two windows: a text window that shows the text being coded, and an event window that shows the events that have been coded from the text. In the default configuration, the text window is larger and placed above the event window, but both windows can be resized and moved to any place on the screen.

13.3.1 Text Window

This window shows the text that is being coded. There will be a slight pause between the display of the text and the display of the events; this is when the coding is actually being done.

Hitting the '0' (zero) key on the keypad of the Apple "Extended Keyboard" or selecting the **Display/Parsed Display** menu option will color-code the source text according to how KEDS has classified it. For example, actors are displayed in red, verbs are displayed in blue, and words KEDS is ignoring are shown with a line through them. See the notes under the **Display** menu for more detail on the parsed display.

The message

Note: transformation rules were applied to the text before coding

indicates that the coded text was modified by transformation rules; see the Classes, Composite patterns, and Rules chapter for details on the use of rules. If rules have been applied, the text that was actually coded may be very different from the original source text. The parsed display will show the transformed text and the list of the rules applied.

13.3.2 Event Window

The coded events are shown in the event window. Incomplete events – those missing a source or target – are displayed in gray; complete events are in black. If no event is complete, the first one with a source or a target will be in black.

If the event display is longer than the event window, it can be scrolled, field by field, using the horizontal scroll bar at the bottom of the window.

The event window also shows "accuracy statistics": the number events coded and the approximate accuracy. The accuracy is calculated by

$$\text{Accuracy} = \frac{\textit{total machine coded events not corrected using } \textbf{Edit}}{\textit{total events generated by machine coding OR } \textbf{Edit}}$$

If the machine correctly ascertains that the text contains no event, this counts as one correct event.

13.3.3 Scrolling

The large arrows in the lower left and lower right corners of the event window move you forward and back through the file of text records and events. KEDS stores about 30 events before actually writing them – the exact number depends on the size of the text and the number of events generated. When the **Edit/Save** menu option is used, this storage is emptied.

You can use the reverse arrow (lower left corner) to go backwards through this list; the small number inside the arrow shows how many events are stored in the reverse direction. Similarly, if you have gone backward, a number will appear inside the forward arrow (lower right corner) showing how many events remain stored in the forward direction until a new event is read. If the reverse arrow disappears, you are at the end of the stored list. You can also scroll through the list using the <**Page Up**> or <**Return**> keys to go forward and the <**Page Down**> or <**Backspace/Delete**> keys to go back.

When scrolling forward and back through events that have already been coded, no recoding is done: this means that any changes made through the **Edit** dialog are preserved. If you want to recode, use the **Recode** menu option, or edit the individual events.

Once an event has been written, you can no longer edit or review it. All stored events are written during a **Save** and when you quit the program.

13.4 Editing Events

Individual events can be edited by clicking on the event record in the event window. This action brings up a dialog that allows the source, target and codes to be modified, as well as allowing the status of the event to be changed. The events written to the

.events files reflect these changes, so the program can be used for machine-assisted coding.

The text of the codes is changed in the usual fashion for Macintosh dialogs: click in the text box and type the appropriate changes. The text edit boxes for these codes are formatted for 6-character codes but a full 12-character code can be entered or edited by scrolling horizontally – select the area at the right end of the box and move the mouse a bit further to the right. If the source and target need to be reversed, use the **Swap** button in the dialog. When an event is edited, the Not an Event status is switched to off – if you are editing the event, presumably you want to save it – though this can be reversed manually.

> **Note:** The strings entered in the edit boxes are truncated to the lengths specified in the display formats. For example, if the Target code is specified as 4 character and you enter 'ABCDEF', only the four characters ABCD will be displayed. If the Edit dialog doesn't seem to be accepting your changes, check the **Display** formatting

The **New Event** option in the **Modify** menu creates a new event template for doing machine-assisted coding and automatically opens the event editor.

The **Notes** field will take up to 16 characters; see the **Notes** section in the Formatting Output chapter concerning how this field is written to the output file. The **Notes** field can be used in machine-assisted coding to add manually-coded information (e.g. context or issue information) to the event record.

Incomplete events – those missing a source, target or actor – are "dimmed" on the screen. The **Not an Event** check box can be used to eliminate any incorrect events. The missing event code *** is used as a place-holder for missing information.

13.4.1 Editing using the keyboard

The event editing dialog can also be invoked using the keypad keys on the Apple extended keyboard: **1** edits first event, **2** edits second event and so forth. The numbers correspond to the complete list of events, not just those displayed in the window.

The following keys modify the effect of a click:

Clicking the event while holding down the command key (the apple key) will toggle the **Not an event** status without displaying the dialog.

Clicking the event while holding down the option key will reverse the source and target.

Clicking the event while holding down the control key will create a new event that is a duplicate of the event that was clicked.

The command and option buttons also operate with the keypadfor example **command-Keypad-2** will change the **Not an event** status of the second event.

By combining these operations, all event editing can be done without the mouse. Use the keypad to select the event to be edited; enter information into the **Notes** field,

move between the source, target, event and other fields using the <**tab**> key, and use <Return> to close the dialog. Use <**return**> in the main window to go to the next event.

13.5 Summary: Keyboard Shortcuts

command + click	Toggles **Not an Event** status
Option+click	Reverses source and target
Control+click	Make a duplicate of the event
Keypad keys 1-9	Same effect as clicking event n
Keypad 0	Toggle between source text and parsed text display
<**return**>, <**enter**>, <**page down**>, <**home**>	Forward one record
<**backspace**>, <**delete**>, <**page up**>	Go back one event

Chapter 14

KEDS MENUS

14.1 */HELP

All of the menus contain **Help** options that describe the basic functions of the menu. These are substantially less detailed than this manual but may be of some use to individuals who don't read manuals.

14.2 FILE

14.2.1 Open Text File

Closes the current input *.Text* file, after writing all coded events to the *.events* and *.text* files, then gives a file selection dialog for selecting a new input *.Text* file. The new file should be in the same folder as the earlier file and the dictionaries. The event statistics are also reset to zero.

> **Note:** The current file is not actually closed until the new file is selected, so selecting **Cancel** in the file selection dialog cancels the menu selection without doing anything.

14.2.2 Save Dictionaries

Saves any changes to the *.Actors, .Verbs, Class*, then returns to the program. During a **Save**, and during a **Quit**, the program does the following:

1. All coded events are written to the *.events* and *.text* files. If a *.text* file is being written, the text saved will include any changes made using the **Modify/Text** dialog.

2. The *.Verbs* and *.Actors* files are written out in alphabetical order and in uppercase, rather than the order that they were originally entered.
3. The *.Verbs* file is reformatted to line up the phrase codes.
4. Comments in the files are preserved verbatim, though occasionally long comments on long phrases might be truncated.
5. Any lines after the ~ ~ **FINISH** statement are also copied verbatim.
6. The coder and date are added to any new actors, verbs or verb phrases.
7. If classes or rules are being used, the classes in the *.Class* file are written in alphabetical order by class name, and the standard classes <CONJ>, <PREP>, <NUMBER>, <PRONOUN>, <PLURAL>, and <TITLE> are included in this file. Comments in the original *.Classes* file are *not* carried forward to the new file. The <NUMBER> class will also include any words that were designated as numbers by numeric issues.
8. The old versions of the files are saved as *File_Name.backup.*

The *.Options* file must be changed externally and cannot be updated from inside the program.

A **Save** purges all of the events from memory as it writes them to the *.event* and *.text* files, so the reverse scrolling arrows will not go back to events that were coded prior to the **Save**.

> **Note:** You will be automatically prompted to save files after a specific number of adds, deletes and changes are done using the **Modify** menu. The number of changes made between prompts can be changed using the **SAVE** command in the *.Options* file.

14.2.3 Quit

Terminates program, closing all output files (*.events, .probs, .complex, .text*), as well as *optionally* saving changes to the *.Actors* and *.Verbs* files. If the *.events* or *.probs* files are empty, they are deleted from the disk.

14.3 MODIFY

The **Modify** menu is used to modify the text, the *.Actors, .Verbs* and *.Classes* dictionaries or add new, blank events. Because the various **Modify** dialog need to manage the entire list of verbs, actors, classes and rules, making modifications using the dialogs is somewhat slow. When an extensive set of changes needs to be made – for example when creating a new coding category using existing vocabulary – it is faster to edit the files using an ASCII editor.

After you make a number of changes – the default is 8 – you will automatically be prompted to save the changes. You can ignore this request, and the number of changes

made between prompts can be changed using the **SAVE** command in the *.Options* file.

If the **LOCK DICTIONARIES** command has been used in the *.Options* file, the **Modify/Actors, Verbs, Text, Classes** and **Rules** selections are disabled and cannot be used.

> **Note:** In the default window configuration – which was the developed for the small screens of the classic Macintosh computers – the **Modify/Verb, Modify/Actor** and **Modify/Phrases** dialogs cover the source and event windows, obscuring the text you may wish to enter. The dialog windows cannot be moved, but on a large screen, the text window can be resized and moved below the dialog so that the text is visible while you are making modifications.

14.3.1 Recode

This option recodes the current record using any changes that have been made in the actors, verbs, classes, rules or text. This is only required when you want to recode an earlier event using dictionary changes made after that event was originally coded; the current record is recoded automatically whenever one of the **Modify** dialogs is used.

14.3.2 New event

This creates a template for a new event and open the event editor. This option is used in machine-assisted coding. If the text satisfied the complexity conditions, New Event overrides this.

> **Note:** If some of the information you need to enter is in an existing event, use the **Control**+click (or **Control**+keypad) to make a copy of that event, then edit the new fields.

14.3.3 Verbs

Changes the verbs and phrases in the *.Verbs* file. To change a verb, scroll to the appropriate entry and click it. The verb and its code will appear in the edit boxes; type the <**tab**> key to edit these fields.

When the **Modify/Verbs** dialog is first entered, and when the verb list is selected by clicking any entry inside it, a border appears around the list and typing alphabetical keys (**A** through **Z** will automatically scroll to the first entry beginning with those letters). The keyed-scroll is reset whenever you click again in the list (including its scroll bar) When the <**tab**> key has been pressed, or one of the text edit boxes or command buttons is selected, alphabetical keys cause text to be entered.

To save the changes in memory, click the **Add, Change** or **Delete** buttons (or use the **command-A, command-C** or **command-D** key combinations). If you have not entered a code, the program will ask for one; the program also presents a dialog verifying any delete operations.

To edit a phrase, click the **Edit Phrases** button, use the **command-E** keys or double-click a verb. This action will produce a list of the phrases associated with the verb; these are edited in the same manner as the verbs. Alphabetical scrolling does not work with the phrases.

> **Note:** If you have previously copied text in the **Modify/Text** dialog (see below), that text can be pasted into the verb or verb phrase box using **command-V**. This allows words to be copied directly from the text into the dictionaries.

Changes in the verb list are made immediately in memory, but are not changed in the file until either **Save** or **Quit** are selected. You will be given the option of leaving the program without saving changes.

The dialog ends when you click **Done** or type the <**Return**> key. The **Modify/Verb** dialog causes an automatic **Recode** of the event; this is necessary in order to insure that the references to verbs, codes and so forth are accurate.

14.3.4 Actors

This dialog changes the *.Actor* file and works in the same fashion as **Modify/Verb**: Scroll to the appropriate entry, double-click it and make the changes. The alphebetical scroll functions just as it does in **Modify/Verb**. As with **Modify/Verb**, the dialog closes when the **DONE** button is clicked or the <**Return**> key is typed; this option also causes an automatic **Recode**.

Date-restricted Codes

If a code involves date restrictions, it will appear inside "[ ]" brackets, and new codes should also be entered with brackets. For example, to add a date restriction to the code

GEORGE BUSH

USA:GOV

enter

[USA:GOV <930120]

and then click the **Change** button.

14.3.5 Text

This menu item invokes a simple editor that allows changes to be made in the text record; it is particularly useful if the text is going to be saved for possible recoding at a later date. This editor is not intended to be used for making extensive changes – use a text editor for that purpose – but it is useful for minor changes and annotations. The editor can also be used to cut-and-paste text to the **Modify** dialogs.

The editor works with one line of the text at a time; click the **Next** and **Previous** buttons to scroll between lines. Up to 241 characters can be included on a single edited "line"[1]; a blank line is appended to the text for adding additional material. The first line displayed will be the first line of the English-language text; click **Previous** to edit the line containing the date and ID information.

If the **OK** button is clicked, the changes will be incorporated into the source text and the record will be recoded. The text is reformatted into 80-character lines broken at spaces. These changes will also be made in the *.event, .text*, and *.complex* output files if the text is being written to any of those files. However, the changes are **not** made in the original *input text* file. If the **Cancel** button is clicked, no changes are made. To copy text, select the text in the edit box, then click the **Copy** button or type **command-C**. This text can be pasted into the text box in the **Modify/Actors** or **Modify/Verbs** dialog using the standard "paste" key combination **command-V**.

14.3.6 Classes

This option allows the membership of classes to be changed. The first dialog will show a list of the classes; double-click the class you wish to modify. The next dialog will list the members of that class. To delete a word from the class, click on it, then click the **Delete** button or use **command-D**. To add a word to the class, type the string into the edit box, then click the **Add** button or use **command-A**. The dialog closes when the **Done** button is clicked or the <**Return**> key is typed; this option also causes an automatic **Recode**.

When you attempt to add a word to a class, the system first searches the existing dictionaries to see whether the word exists in another class, and notifies you if it does.

The **Modify/Classes** option is available only if classes have already been initialized using the **Classes** command in the *.Options* file. Only existing classes can be modified; it is not possible at present to add an entirely new class from inside the program.

> **Note:** The **Modify/Classes** option can add words to all of the standard classes except <ACTOR>, <AGENT>, <VERB>, and <COMMA>. Words from those classes can be added to *another* class, but you can't add into those classes using this dialog

[1] In other words, this is working within the constraints of editable text in a Macintosh dialog text edit box. At some point we may put a full-fledged text editor into the program, but at least this is functional. If the 241 character limit is a problem, do some of the editing, click OK, then use the editor again: leaving the editor reformats the text into 80-character lines and one can then add additional information.

Number Class

The **Modify/Classes** option works slightly differently for the <NUMBER> class. The words in this class are displayed with their default numerical values (if any) following the word in square brackets. The default value can be changed by clicking the word, editing the value, and then clicking the **Add** button (in other words, **Add** acts as a "change" command for the numerical values). The class list includes all of the words that occur in the lists of numeric issues, though the numeric values of those lists are not listed and cannot be edited.

Because the <NUMBER> class automatically includes all words that are being used by numeric issues, a word in that class cannot be deleted if it is being used by an issue – in other words, the only words you can delete are those which were initialized using the **Classes** command in the *.Options* file. If you try to do this, you will get a caution message that lists the first issue where the word is used.

A word also cannot be added to the <NUMBER> class if it is already assigned to one of the standard classes (<ACTOR>, <VERB>, etc). If you try to do this, you will get a caution alert and the word will not be added.

14.3.7 Rules

This option modifies rules: scroll to the appropriate entry, double-click it and make the changes. The choice of the iteration operators $->$ versus $=>$ is made using the buttons in the lower-left corner of the dialog. The dialog closes when the **Done** button is clicked or the <**Return**> key is typed; this option causes an automatic **Recode**.

The **Modify/Rules** option is available only if some rules have already been initialized using the **Classes** command in the *.Options* file.

14.3.8 Comments

This option is only active if a **PROBLEMS** command in the *.Options* file has been used; it brings up a submenu for entering information in the *.probs* file.

Four options are available. Each option writes the original text, the events coded, the word list and some additional information.

Remarks:	Brings up a dialog box for entering remarks in the problem file.
Internal Event:	Inserts a message that the event was internal (i.e. not international).
Actors Reversed:	Inserts a message that the actors were reversed.
Wrong Event:	Inserts a message that the wrong event was coded.

14.4 DISPLAY

14.4.1 Word list

This dialog shows the words recognized by the system and how they were classified; from this list one can usually figure out why the system made an event assignment. This list supplements the information shown in the **Show Parsed Text** display option by showing codes and complete phrases for roots containing multiple, non-consecutive words. The code list is a window and can be moved on a large screen.

> **Note:** If the system appears to be missing its identification of a word, check the spelling in the source text. Spelling errors are surprisingly common in Reuters; one should also include both United States and British spellings (e.g. ORGANIZATION and ORGANISATION) or else just use a stem (e.g. ORGANI)

The reduction of compound actors and titles often results in a word that will not fit in the first column of the list; this is indicated by a "..." at the end of the displayed text. Double-clicking such an entry will display the full text below the word list. The list window closes when the **OK** button is clicked or any key is pressed.

By default, the word list shows only the words that have been assigned class that is used in the coding; clicking the **Show only classed words** check box will toggle this. The word list does not show any words of type STOP , eliminated by **OMIT** delimiters or comma-delimited clauses. If a word appears in the original text and is not in the list, it was eliminated by the parser at some point. The word list will show the original text for compound actors, titled actors and dereferenced pronouns.

14.4.2 Code list

This provides a list of all of the codes in the system along with any text associated with the code in a **CODE** command in the *.Options* file. Event codes, actor codes, or all codes can be listed; the default is event codes. The codes are presented in alphabetical order. The code list is a window and can be moved on a large screen.

This option is not particularly efficient and in machine-assisted coding it is more advisable to provide coder with a xeroxed page containing the codes.

14.4.3 Index

This creates an alphabetical index, by code, of all of the actors, agents[2], verbs and verb phrases in the system. The index is written to a text file $< project >$ *.index*. In the verb phrases, the root verb is shown following the * token.

[2] A separate AGENTS index is written only if the
SET: CODE AGENTS=TRUE
command has been used. This index will only show the explicit agent codes assigned with an AGENT command. Implicit agent codes such as GOV in

The index includes all of the codes assigned to an entry, including null, discard, compound and date-restricted codes. When an entry has a composite code, the entire code string is written after the entry.

Verb phrase diagnostics

The index automatically generates two diagnostics for verb phrases; these are evaluated only if both the verb root and the phrase have simple codes. The diagnostics are merely advisory and can be ignored.

/* Phrase code identical to root code */

This occurs when the code assigned to the phrase is identical to the code of the root. The phrase is redundant because the root alone is sufficient to assign the code.

/* % token without compound code */

This diagnostic is shown when a phrase contains a compound actor token % and no source ($) or target (+) token. A compound code (e.g. -% *PLAN TO MEET [082:082]) is usually appropriate in this situation.

14.4.4 Status

Memory

This displays the amount of memory available in the various storage arrays of the program.[3]

Roots	= number of actors and verbs
Phrases	= number of phrases and issue strings
Word text	= total characters used for actors, verbs, classes, issues and code text;
Phrase text	= total characters used for phrases;
Codes	= total distinct codes

> **Note:** If you run out of memory, the program will *usually* terminate using the standard exit procedure, so you will have the opportunity to

GEORGE_BUSH[USA:GOV]
will be listed in the ACTORS index; this is done whether or not the CODE AGENTS option has been activated.

[3]The use of arrays rather than dynamically allocated lists or trees is due to considerations of speed and memory management: the Macintosh prefers to deal with large blocks of contiguous memory rather than large numbers of pointers. The downside of this arrangement is that storage capacity is fixed for various elements of the program.

save verb and actor changes. KEDS provides warnings when it exceeds 95% of the capacity of its memory buffers; at this point it is a good idea to quit the program rather than pushing it to the limits of memory. KEDS's memory-management routines are not particularly sophisticated and the probability of the program unpredictably crashing under low-memory conditions is decidedly greater than zero.

Word Count

This provides a count of the number of actors, agents, verbs, phrases, and prepositions in the working vocabulary. Unlike the **Memory** report, it counts the number of distinct items rather than showing the amount of memory used by those items.

Project

This displays the information recorded in a project file. When this option is selected, you will be presented with a file selection dialog listing KEDS project files. Project files can be examined even if their associated data files have been misplaced; the project file currently in use can also be examined. The program then displays the basic information about the file, similar to that displayed on the introductory screen. Subsequent screens display the statistics for up to 64 coding sessions. The **Write** button writes the coding session information to a tab-delimited text file; this can be read into a word processor or a spreadsheet.

Information reported:

Date	=date of coding session
Start	=time session started
End	=time session ended
Coder	=coder identification
Texts	=number of text records that were coded
Events	=number of events produced
Accuracy	=percentage of events produced that were accepted without editing

14.4.5 Show Parsed Text

This toggles the type of display in the text window. The parsed text display shows the text as "seen" by KEDS: stop words are not shown; text that is between **OMIT** delimiters or in comma-delimited phrases is shown as deleted, compounds are combined, pronouns are dereferenced, and words are color-coded by type:

Blue	Verb with code assigned to root
Blue, italic	Verb with null-coded root; this will only be coded if one of its phrases is matched
Red	Actor
Red, italic	Implicit actor: agent that was converted to an actor
Green	Agent
Magenta	Dereferenced pronoun
Yellow	Unreferenced pronoun
Cyan	Conjunction
Cyan, italic	Preposition
Black	Unclassified words
Line through word	Word is not being used in the coding because it is a null-coded actor (red), null-coded verb without phrases (blue) is inside a comma-delimited nonrest-rictive phrase, or has been incorporated into a compound actor.

If a root contains multiple words connected by underscores, all of the words in the root will be colored. If the multiple words are not connected by underscores, only the first word will be colored. Use the Display/Word List menu option to see the complete root.

If a word is followed by one or more subscripts, it is a member of a class other than the classes indicated by color-coding. The numbers correspond to the class indices; a list of these is shown following the legend.

Dereferenced pronouns are followed by the actor they refer to. The symbol "– >" indicates that the reference is within the current text; the symbol ">>>" indicates that the reference was forwarded from the previous text.

If transformation rules were applied to the text, a list of those rules is provided in the parsed display. The number in the first column reports how many times the rule was applied; the second column shows the rule itself. The message (additional rules were used beyond those listed here) means that there was insufficient room available to store the list that reports all of the rules used; this will only occur under unusual circumstances.[4]

Note: If there is a phrase containing a discard or complex code in the

[4]This list shares storage with the class assignments, and will only occur when there is a very large number of class assignments and a number of different rules are applied. This has no effect on the application of the rules, only on the report shown on the screen.

sentence, the parsed display will only show a message to that effect: KEDS stops parsing as soon as it finds a discard or complex phrase.

If the parsed list contains garbage, try doing a **Recode** to clean it up. This was a problem in earlier versions of the program but should occur only rarely in this version.

14.4.6 Show Invalid Events

When this option is checked, invalid events (see **Options/Valid Events**) are displayed in gray ("dimmed"). When it is not checked, invalid events are not displayed at all. If there are no valid events to be displayed, the message No valid events is shown. In rare circumstances, this message will also appear if one is scrolling the event window past the number of valid events. Selecting this menu item toggles the checkmark.

14.5 Options

14.5.1 Valid Events

This dialog sets the conditions for whether an event is considered valid and written to the output file; it is the menu equivalent of the **VALID** command in the *.Options* file. The default condition requires both a source and a target; either of these requirements can be changed using this dialog (or using the **VALID** command in the *.Options* file). Checking a **Source** or **Target** option is equivalent to putting in the VALID command.

The validity condition is true if an event has an non-null code for the actor (source or target). If agents are being coded, either the actor or the agent must be non-null.

The settings in this option are saved in the project file and set to those values the next time the project is used unless overridden by a **VALID** command in the *.Options* file.

14.5.2 Pause When

This dialog sets the conditions under which KEDS pauses before coding the next record, giving you the option of recoding. It is the menu equivalent of the **PAUSE** command in the *.Options* file. This is useful is one is dealing with source material that contains irrelevant material such as purely internal events, when you only want to look at the text when KEDS has not found any events, or when you only want to look at complex text.

The three code-related options are:

Always

KEDS pauses after coding each event; this is the default.

Contains

Pause when the coded text contains *all* of the checked items. In machine assisted coding, this option is typically used as a filter to bypass sentences that contain nothing of interest. Note that if you are only interested in verb phrases ("events") and not actors (for example if you are using KEDS to code something other than event data...), sources and targets can be set to default values.

Missing

Pause if the coded text is missing *any* of the checked items. This option is typically used in machine assisted coding to fill in missing information.

The **Source, Target** and **Event** items are selected using check boxes. Note that **Contains** acts as a logical AND while **Missing** acts as a logical OR in this selection.

If the input contains a discard phrase, it is always skipped if the **Contains** or **Missing** options are active.

Checking the **Complex** option causes the program to pause whenever complex text is found; this acts as a logical OR with the code-related options.

The settings in this option are saved in the project file and reset to those values unless overridden by a **PAUSE** command in the *.Options* file.

14.5.3 Complexity

This dialog sets the complexity conditions (see Complexity Filter section in the .Options File chapter). The logical conditions are selected using check boxes. Numerical options are activated by setting them to a non-zero value. If a non-numerical value is entered for a numerical option, the system will beep after **OK** is checked and the option set to zero. A **Recode** is done automatically after the dialog is closed.

If you want to divert complex sentences to a *.complex* file during autocoding, activate at least one of the complexity conditions using the **COMPLEX** command in the *.Options* file. Any changes in the complexity conditions made through this menu option will be reflected in the contents of the file.

> **Note:** because KEDS initializes all of its required files after the *.Options* file is read, a *.complex* file cannot be activated using the menu alone.

14.5.4 Parameters

This dialog allows many of the *SET:* parameters to be changed temporarily; these changes will not be reflected in the *.Options* file. A recode is done automatically after the dialog is closed.

The parameters associated with **CODE AGENTS** cannot be changed unless the **SET: CODE AGENTS=TRUE** was included in the *.Options* file. If a non-numerical value is entered in the **Look Forward** or **Look Backward** box, the system will beep after OK is checked and the value will not be changed.

14.5.5 Auto Coding

Selecting this option puts the program into automatic coding mode: it will continually code events without pausing and without stopping to allow editing. It is typically used to code a set of data using an established dictionary. In autocoding mode, the program does not write events unless a source, target and event have been identified.

To stop the automatic coding, select this option again from the menu bar. In each case, you will be asked via a dialog box to verify the command. When autocoding is stopped, the dialog box shows the number of events coded and the time required for the coding.

To *temporarily* stop automatic coding, hold down the mouse button while the cursor is in the text window. Coding will resume when you release the button.

If the **Enable Multifinder** option is checked, KEDS's autocoding will run as a "background" process in Multifinder: you can switch to another program and KEDS will code at the rate of about two events per second during any idle time available. However, enabling Multifinder slows the autocoding substantially even when you aren't using another program – for example it halves the coding speed on a Mac 7100 – so unless you intend to use your machine for another task, leave this option unchecked.[5]

> **Note:** If Multifinder is enabled, you may want to disable any screen-savers when doing autocoding. KEDS will autocode while a screen-saver is running but slows to the background rate of two events per second because it thinks the screensaver is another program. High cost for watching flying toasters. When the Multifinder option is disabled, the screensaver will be activated after the program has finished coding so you can safely leave the program alone after starting the autocoding.

See Also: The **INPUT FILES; Text File; Coding a number of files** section for instructions on setting up the *.Text* file to autocode a set of input files.

14.5.6 Skip Records

This option provides two ways of skipping ahead in the input file. When skipping, the records are not coded and they cannot be accessed using the reverse arrow.

Until date $\geq$ date

Enter the date to which you want to skip in YYMMDD format (e.g. 950809= 9 August 1995). Records will be skipped until the first record *after* a record with a date that is greater than or equal to that date. If you want to stop exactly on the first record of

[5] The situation is actually a bit more complicated than this: even with the Multifinder option disabled, you can still switch to another program by selecting it from the Multifinder icon (right side of the menu bar in System 7), then clicking a second time. This will allow you to operate the alternative program, but no background coding will occur. This is a quirk rather than a deliberate feature and may not be maintained in future versions of KEDS or the Finder.

the date, skip until the previous day and then manually go through the records of the previous day.

Next number records

Enter the number of records you wish to skip. If this is greater than the number of records in the file, the program will beep and display a blank text record when it reaches the end of the file; hit <**Return**> to exit. If you do not enter a valid number, the option will do nothing.

Appendix A

PROGRAM LIMITS

Note: This is only a partial list of the maximum limits

Lines in a text record	32
Words in a text record	255
Characters in a text record (including blanks)	2560
Characters in IDInfo string in header (default=6)	15
Value of story and sentence sequence numbers	32767
Number of codes in a string	64
Number of distinct codes in system	1023
Characters in a verb phrase	80
Sets of OMIT delimiters added in the *.Options* file	3
Sessions tracked in Project file	64
Events found in a single text (after compound expansion)	32
Total characters in verb, actor and agent roots, and issue strings	131072

Total distinct verb, actor and agent roots	8192
Total characters in phrase patterns	262144
Total distinct phrase patterns	16380
Past events stored	32
Sentence conjunctions tracked	8
Number of stop words	64
Number of issues	10
Number of elements in OUTPUT, DISPLAY or LABELS format	32

Appendix B

ERROR MESSAGES

B.1 Program reports garbage in input files

The program may report garbage in input files that appear okay in a word processor or ignores *.Options* commands.

Be sure that the files were saved as **Text Only**: Microsoft *Word* for example occasionally decides that you want a file saved in Word's **Normal** (i.e. formatted) format even though it was originally saved as **Text** (pasting something from a **Normal**-formatted file seems to do this, for example). The formatted files contain a large amount of non-text material prior to the text: when the *.Actors, .Verbs* and *.Class* files are read this incorrect file format will immediately be obvious, but routine reading the *.Options* file will usually just skip over this information. Under some configurations (notably "Fast Save") *MS-Word* does not insert new material into the file where you expect, and KEDS will appear to be ignoring the new commands. To correct the problem, use **Save As...** to re-save (i.e. replace) the file under the same name with the **Text Only** file format option selected.

B.2 Error messages during input phrase:

> **Note:** If the program encounters any of these errors while reading the *.Actors, .Verbs* or *.Options* phrase lists, you will be given the choice of continuing to check the phrases, but the program will not run unless the errors deal only with pattern syntax errors. Make the appropriate corrections using a word processor and then re-run the program.

- *Blank line*
  - **Meaning:** A blank line has been encountered when a phrase was expected

  - **Solution:** Eliminate line.

- *Phrase has actor token connected with underscore*
  Phrase is missing a verb token (*)

  - **Meaning:** See comments below under Editing

- *No code ("[...]") has not been specified*

  - **Meaning:** The code is missing for an actor, agent, verb, or phrase.

  - **Solution:** Add the code.

- *No phrase or pattern has been specified*

  - **Meaning:** The first item encountered in an input line was the code. This usually occurs when <Return> was accidentally hit after entering a phrase or pattern and will be preceded by a missing code error in the previous line.

  - **Solution:** Add the phrase or pattern, and remove the <Return>.

- "*< char >*" *or* "*< char >*" *cannot be indexed*

  - **Meaning:** The first two characters of a phrases can only contain characters between '@' and '_' in the ASCII sequence; for practical purposes, this means upper-case letters. Lower-case letters are automatically converted to capitals and a few diacriticals are converted, but other characters (for example, '$,' which is KEDS's token for the location of the event source) will cause problems.

  - **Solution:** Eliminate the characters causing the problem.

- *The phrase "..." has already been entered as a [word type]. The second entry will be ignored*

  - **Meaning:** The same phrase has been entered more than once, so the entry is either an unintentional duplicate or the function of the word is ambiguous. If the phrase is a verb, the phrases associated with the verb are also eliminated.

  - **Solution:** Eliminate the duplicate phrase in the appropriate file. Alternatively, if you are certain that the second entry is the one you wish to eliminate, just save the file, since the version of the dictionary in memory will not contain the duplicate. Hopelessly ambiguous words such as **GEORGIA** may need to be assigned a complex or discard code.

- *Error in NOTES format:*
  ...
  This produces a string longer than 255 characters; the format specification has

been truncated.

  - **Meaning:** A replication factor (“ ^ nnn^ ”) in a NOTES format produces a format string that is too long. This can occur either because the replication numbers are two great (e.g. [ABC ^ 156 ^ EFG ^ 100 ^] or because you forgot the closing “ ^ ” in the specification.

  - **Solution:** Shorten the replication factor or add the closing ^ .

- *The following pattern has a syntax error:*
  < pattern >
  △
  Problem: < problem description >
  The pattern will be ignored.

  - **Meaning:** A syntax error in a pattern. The △ points to the approximate location of the error.

- **Syntax errors detected:**
  - *Pattern is missing a verb token “*”*

    * All verb phrases must contain a * indicating the location of the verb
  - *Verb token “*” can only be preceded/followed by _ or blank*

    * Text (e.g. suffixes or verb endings) cannot come directly before or after the verb token; these must be entered as separate words.
  - *“_” cannot follow a blank*
  - *“_” following “|” or “{”*

    * An underscore can only follow a string, *, +, $, % or }
  - *No closing “>” in class name*

    * ‘<’ without a corresponding ‘>’
  - < string > *is not a valid class name*
    string has not been declared as a class; check for misspelling
  - *A “ ~ ” must be followed by a “{”*

    * A set { } must follow immediately after any not operator ~
  - *Missing “}”*

    * There are insufficient closing brackets
  - *Too many “}”*

    * There are too many closing brackets

  - *No format continuation, add ^ ^ to beginning line*
    * No continuation marker ^ ^ at the beginning of an extended format command
  - *ISSUE command; Missing ‘ in ’ -delimited string*
    * The ISSUE command expects an apostrophe to be the first non-blank character after the equal sign in its components. If none is found, it returns a null string for that component.
  - *ISSUE file* $< file - name >$*;Non-numeric priority; value set to zero*
    * The characters inside an issue code priority cannot be interpreted as a number. The issue and its code will be maintained but the numerical priority will be set to zero,
  - *Numeric ISSUE file* $< file - name >$*;Non-numeric value string.*
    * The characters inside a numerical value cannot be interpreted as a number. The numerical issue word will be ignored,
  - *ISSUE command; Unrecognized DEFAULT variable*
    * The string following @ in the DEFAULT= field does not correspond to a legal variable name. The default is ignored but the issue is still coded.
  - *ISSUE command No TAG string set in numeric issue*
    * If the OPTIONS=‘NUMERIC’ field has been used to specify a numeric issue, a TAG=‘*string*’ field must also be used to specify the tag string,
  - **Solution:** Correct the error

B.3 Error messages in Modify dialogs

- *Phrase is missing a verb token (*)*
  - **Situation:** Editing verb phrases
  - **Meaning:** A verb phrase must contain a “*” character indicating the location of the verb in the phrase.
  - **Solution:** Add a “*” at the appropriate location in the phrase.
- *Please enter a code before adding*
  - **Situation:** Editing verb phrases
  - **Meaning:** A phrase has been entered but there is no code associated with the phrase.

  - **Solution:** Enter a code (click in the **CODE** box or use the tab key); if you don't want any code associated with the phrase, use the null code - - -.

- *< word > is being used by ISSUEn and cannot be deleted.*
  - **Situation:** Attempting to delete a word in the <number> class.
  - **Meaning:** The word is being used at least one numeric issue (the message displays the name of the first issue that used the word) and therefore cannot be eliminated from the <number> class.
  - **Solution:** Eliminate the word from the issue(s). If only the default value is problematic, set it to zero by editing and using **ADD**.

- *< word > has already been assigned to a non-numeric standard class; it cannot be changed to a number.*
  - **Situation:** Attempting to add a word to the <number> class.
  - **Meaning:** Words in the <number> class are assigned a primary type of <number>. Words already assigned to one of the standard classes – for example actors, verbs, agents or conjunctions – cannot also be numbers. A word that is in a non-standard class can also be a number
  - **Solution:** None.

B.4 Error messages relating to memory

- *Out of memory while trying to initialize the < array name > array. Click OK to terminate this run, then double-click the new project file to start over: program will probably work the second time.*
  - **Situation:** Usually after project initialization
  - **Meaning:** Possibly the total size of the actors, verbs and issues lists is too long, but this occurs sporatically and we're not really sure....
  - **Solution:** As noted, exiting the program (which may crash anyway) and then restarting it will usually solve the problem. You've reached the capacity of the program; shortening the lists might also help. If you have reduced the memory size below the recommended level, put it back to the recommended size, and raising it above the recommended size might also be helpful. This error is due to some sort of initialization bug that we are trying to track down.

- *Memory manager error at < handle name > in < procedure name >. This indicates an uncorrected program bug: please contact p-schrodt@ukans.edu for an update. Click OK to terminate program.*
  - **Situation:** Initialization and after dialogs
  - **Meaning:** This indicates a bug in the program.
  - **Solution:** None: if the program is working correctly you'll never see this. If you do get the message and are in luck, it will have been solved in a later version of the program. If not, we can probably use your vocabulary lists to track it down.

- *Out of memory while trying to initialize the <array name> array in <procedure name>. The <description of KEDS function> procedure will not be run. Click OK to continue.*
  - **Situation:** Procedures that require additional memory but can be bypassed without crashing the program, e.g. rule implementation and project reports
  - **Meaning:** The total size of the actors, verbs and issues lists is too long for the available memory.
  - **Solution:** You've reached the capacity of the program; shorten the lists or increase the allocated memory. The program will continue to function – save your lists! – but the frequency of this error message will probably begin to annoy you.
- *Click OK to terminate program. You are running low on memory in < location >. Adding actors, verbs or phrases may cause a fatal memory error. Save your actors and verbs lists, quit the program, and edit those lists to make them shorter. This is only a warning*
  - **Situation:** Reading or editing actors, verbs or verb phrases
  - **Meaning:** The total size of the actors, verbs and issues lists is too long.
  - **Solution:** You've reached the capacity of the program; shorten the lists. If you are really desperate, you can free up some memory by eliminating the CODE statements in the *.OPTIONS* file, but generally you must eliminate phrases. This warning occurs when you've got about 5% of the memory remaining and allows for a graceful exit; if you ignore it and actually run out of memory, the program will probably crash in the near future. If you are near the limits of memory, it is advisable to do the editing using a word processor rather than in KEDS
- *The following memory problem has occurred:*
  < explanation >
  Click OK to terminate program; changes will be lost. Increasing the application memory size MAY solve the problem.
  - **Situation:** Reading or editing actors, verbs or verb phrases
  - **Meaning:** Occurs if phrases continue to be added after earlier warning, but now terminates the program.
  - **Solution:** See above. Increasing the memory size usually won't solve the problem but in a few circumstances it will.
- *Insufficient memory to display Help text*
  - **Situation:** When activating any of the Help menu options
  - **Meaning:** There is insufficient available memory to read the Help files into memory and display them. This warning does not affect the program; it just tells you that the help information will not be displayed.
  - **Solution:** Increase the **Preferred size** memory size for the programin the Finder, single-click the program icon and type **command-I** (or select **File/Get Info** in the Finder menu), then increase the Preferred size number in the **Memory** Requirements box.

- *Insufficient space for the VERIFY string < string > in the text buffer; it will be ignored*
  - **Situation:** When running the program in VERIFY mode
  - **Meaning:** There is insufficient available memory to store all of the verification strings and the source text.
  - **Solution:** Decrease the size of the test text or the number of events coded.

B.5 Error messages relating to files

- *KEDS can't find the file <file name or garbage> . Please be sure this is in the appropriate folder. Click OK to quit the program.*
  - **Situation:** Reading during initialization
  - **Meaning:** One of the input files.*Verbs, .Actors, .Classes, .Options,* or a text or issues file isn't where KEDS expects it.
  - **Solution:** This typically occurs after you've rearranged files in different folders, so the subdirectory structure is different than it was when the project file was originally set up. Either restore the arrangement of the files, or reinitialize the project file. You may also have misspelled the name of the file in a CLASS or ISSUE command.
- *KEDS is having difficulties with the file <file name> (File error -34 :Disk Full).*
  - **Situation:** Usually during a file save; also it could occur while coding
  - **Meaning:** Your disk is full.
  - **Solution:** KEDS uses a number of temporary disk files. During a save, KEDS holds three copies of the *.Verbs* and *.Actors* lists: the new copy, the original and the backup. If the save is successful, the backup is deleted and the original renamed as backup; if the save is not successful, you may see a file with the suffix.*backup.2* this is your previous backup. KEDS also creates a file called *KEDS.Temp.1* that holds coder date and identification information from the lists.

    You can also run out of disk space while processing events and writing them to an output file. The coded data may take considerably more room than the original text, particularly if you are saving text records.

    The upshot: don't run KEDS with a disk that is nearly full.
- *KEDS is having difficulties with the file <file name> (File error -47:File open in another application).*
  - **Situation:** Usually during initialization
  - **Meaning:** In all likelihood, you've got one of the input files (.*Verbs, .Actors,* etc.) open in a text editor.
  - **Solution:** Close the file.

- *KEDS is having difficulties with the file <file name> (File error # < error > < description >).*
  - **Situation:** During any disk operation
  - **Meaning:** The Macintosh has returned a file error and there is nothing KEDS can do about it. The file error number is from the standard set of Macintosh disk file errors.
  - **Solution:** Try again; it could just be a random glitch. Restarting the computer also may help. This error should be quite infrequent unless your machine is having mechanical problems.

B.6 Program crashed after initializing a project file

Well, this happens occasionally... Just double-click the new project file to start over; the program will work fine the second time.

B.7 Program skips records without stopping:

Check the settings in the **Options/Pause When** menu command; these may have been changed by a previous user. Selecting the **Always** option will cause the program to stop for each record.

B.8 Program does not produce *.events* files:

Check that the project name is less than 16 characters – if the name is too long it adding the *.events.nn* suffix may produce an illegal file name, so the file will not be saved (a future version of the program will contain an error message for this condition).

B.9 Program isn't writing complex records to the *.complex* file

There are two ways this could happen:

During human-assisted coding: KEDS doesn't write a *.complex* file except during autocoding.

During autocoding: You need to include a **COMPLEX**: statement in the *.Options* file.

B.10 Program is beeping while coding:

This is usually due to an infinite loop in a REPLACE command or a rule. A very composite pattern with a large number of embedded patterns could also cause this; there is a recursion limit of 16 levels in the pattern matching, though it is very unlikely that this limit will be hit unless there is something wrong with the original pattern.

Appendix C

REVISION HISTORY

Version	Date	Features
0.1	91.2	Basic pattern matching for actors and verb phrases. Basic project file procedure and line-oriented user interface.
0.2	91.7	New parsing system with separate lists for actors, verbs, and stopwords; syntactic information used to identify actors and event codes. Interface and documentation expanded; initial work on Macintosh interface. Implemented German-language facilities.
0.3	91.9	Full Macintosh interface; New Event option; compound actors and verbs, pronoun substitution, comma-delimited clause elimination, title-compression, original agents facility.
0.4	92.6	Not released; experimental version used to code sources other than Reuters.
0.5	93.5	Coded compound actors, project status report; Notes facility; keypad Edit options. Considerable debugging and elimination of experimental features in version 0.4.
0.6	94.2	Extended event editor; formatted output; scrolling text and event windows; agents and issues coding; PLACE searching.
0.7	94.11	Enhanced output formatting, compound agents, complexity detection, multiple file output, date restricted actors. PANDA options integrated into main program, German options deactivated.
0.8	95.6	Pause when, Skip records, Word count, and Help options
0.9	95.12	Composite patterns, rules and classes; new manual; complex and discard codes; index; pronoun forwarding; AUTOFILE option; enhancing parsing of compound actors.

Appendix D

EVENT DATA RESEARCH

D.1 Introduction

> **Note:** This appendix is based on Philip A. Schrodt. "Event Data in Foreign Policy Analysis" in Laura Neack, Patrick J. Haney and Jeanne A.K. Hey, eds. *Foreign Policy Analysis: Continuity and Change in Its Second Generation*. New York: Prentice Hall, 1994.

Foreign policy analysis developed at about the same time as the behavioral approach in political science. The objective of the behavioralists was to study political behavior using systematically measured variables, statistical techniques, and unambiguously stated hypotheses. In some areas of political science, the behavioralist studies used measurement techniques that had been developed earlier. For example, researchers attempting to model elections found that the traditional questions asked of potential voters in survey research – party affiliation, whether they had voted before, who they were planning to vote for and so forth – provided a useful foundation for their studies. While the statistical methodologies and survey methods used in contemporary voting research are substantially more sophisticated than the voting surveys of the 1920s and 1930s, the basic measurement instrument – the public opinion survey – is the same.

No equivalent data existed in the field of foreign policy analysis. Traditional studies of foreign policy primarily used narrative sources such as documents, histories, and memoirs and there was no way to directly analyze these in a statistical framework. This disjuncture necessitated the development of new methods for generating data. A variety of these methods have been discussed in the other chapters of this volume; this chapter will focus on one of the most commonly used measurement techniques, event data.

The basis of many studies of foreign policy is the fundamental question of "who did what to whom?" For example, during the Nixon administration (1968-1974), the

United States and the Soviet Union had a relaxation of diplomatic tensions known as the *détente* period. This was reflected in a variety of foreign policy actions, including arms control agreements, a decrease in hostile rhetoric, increased trade, and increased cooperation in resolving disputes. A decision maker living during this period would have a general perception that the hostility between the two superpowers had decreased. However, this perception would be based on a general pattern of cooperative interaction, rather than on a single incident.

Event data are a formal method of measuring the phenomena that contribute to foreign policy perceptions. Event data are generated by examining thousands of newspaper reports on the day to day interactions of nation-states and assigning each reported interaction a numerical score or a categorical code. For example, if two countries sign a trade agreement, that interaction might be assigned a numerical score of +5, whereas if the two countries broke off diplomatic relations, that would be assigned a numerical score of -8. When these reports are averaged over time, they provide a rough indication of the level of cooperation and conflict between the two states.

Figure 1 shows the actions that the United States directed towards the Soviet Union for the period 1948-1978 as measured by the Conflict and Peace Data Bank (COPDAB) event data set collected by the late Edward Azar (1980,1982).[1] In the COPDAB coding scheme, negative numbers indicate conflictual behavior; positive numbers indicate cooperation. COPDAB is based on *The New York Times* and a variety of regional newspaper sources; the data cover the period 1948-1978.

The COPDAB time series shows three general periods. The early Cold War (1948-1962) is characterized by uniformly negative relations, though these are more stable in the late 1950's than in the early 1950's. A partial "thaw" occurs in 1962-1970 following the Cuban Missile Crisis, with the relationship being neutral. Finally, the 1970-1978 period shows the rise and fall of the *détente* policy. Other event data sets covering the 1980s record the "new Cold War" of the early Reagan period followed by the improved relations that occur when Gorbachev comes to power in the USSR.[2]

The event data record of USA-USSR interactions correspond closely to the patterns one would expect from an historical study. Moreover, the event data can also be used to fine-tune that chronology. For example, while Nixon clearly intended to implement a *détente* policy from the beginning of his administration in 1969, there was continued disagreement between the USA and USSR over the US involvement in Vietnam, the 1968 Soviet invasion of Czechoslovakia and other issues, so the interaction pattern is not actually positive until 1971. Positive interactions peak about the time of Nixon's resignation in 1974; the event data scores then decline during the two years of the Ford administration and return to post-Cuban Missile Crisis levels by 1976.

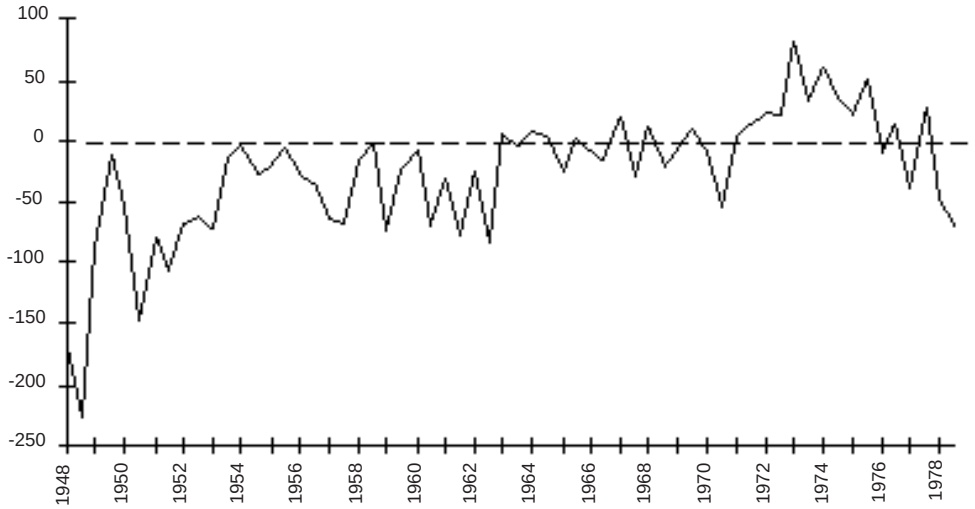
Figure 2 shows another example of the use of event data to chart the evolution of a complex international interaction, the Palestinian *intifada* (uprising) that began in December 1987.[3] This chart is based on the coding of news stories on Israeli-Palestinian reported by the Reuters international news agency. These reports were automatically coded by a specialized computer program into the World Events Inter-

[1] Figure 1 is based on the COPDAB scores reported in Goldstein and Freeman (1990:162).

[2] See Goldstein and Freeman 1990.

[3] The data in Figure 2 and the Israel-Lebanon time series mentioned below are discussed in Schrodt and Gerner (1994).

Figure 1
USA Actions Towards USSR, 1948-1978



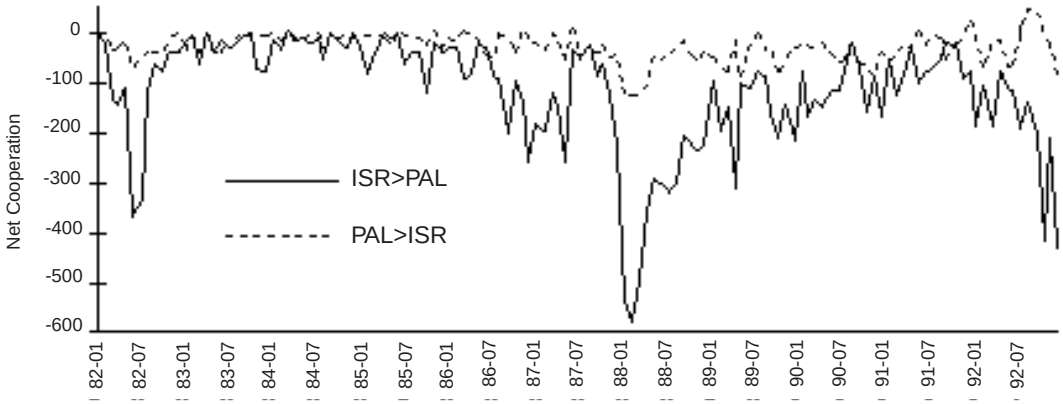
action Survey (WEIS) event data categories developed by Charles McClelland (1976) . The categorical WEIS events were then converted to a monthly numerical score using a scale devised by Goldstein (1992); as in Figure 1, negative scores indicate conflict and positive scores cooperation.

This time series shows the pattern of interactions – largely uses of force – in considerable detail. The initial increase in conflictual activity in 1982-83 corresponds to Israel's invasion of *Lebanon*, which was initially directed against Palestine Liberation Organization forces. The invasion is followed by a period of five years of relative quiet, though a separate series of event data on Israel's interactions with Lebanon during this period shows a great deal of conflict as opposition to Israeli forces shifts from the PLO to various Lebanese groups. The *intifada* begins abruptly in December 1987 and then gradually declines over the next five years, though there is another upsurge in violence following the election of a Labor government in Israel in the summer of 1992.

As with the case of the USA-USSR interactions, this time series gives a more exact measure of the patterns of events over time. For example, while the *intifada* follows a lull in conflict during the summer of 1987, the event data also show a general increase in conflict beginning about 18 months earlier. This increase may have been a precursor to the larger uprising.

As these two figures illustrate, event data can be used to summarize the overall relationship between two countries over time. The patterns shown by event data usually correspond to the narrative summaries of the interactions found in historical sources, but unlike narrative accounts, event data can be subjected to statistical analysis. As a consequence, event data are frequently used to study foreign policy outcomes and some characteristics of the international environment within which foreign policy decisions

Figure 2
Israel-Palestinian interactions, 1982-1992



occur.

D.2 Creating Event Data

The creation of event data is basically a process *content analysis* (see Krippendorf 1980; Weber 1990) and involves three steps First, a source or sources of news about political interactions is identified. This could be an internationally-oriented newspaper such as *The New York Times*, a set of regional newspapers and newsmagazines, a news summary such as *Facts on File* or *Deadline Data on World Affairs*, or a newswire service such as Reuters or the Associated Press. As will be discussed below, the choice of the event source can have a substantial effect on the number and type of events reported.

Second, a coding system is developed, or a researcher may decide to use an existing coding system such as WEIS or COPDAB. The coding system specifies what types of political interactions constitute an "event," identifies the political actors that will be coded (for example, whether nonstate actors such as international organizations and guerrilla movements will be included in the data set), specifies the categories of events and their codes, and specifies any information to be coded in addition to the basic event. For example, the COPDAB data set codes a general "issue area" whether an action is primarily military, economic, diplomatic or one of five other types of relationship. WEIS, in contrast, codes for specific "issue arenas" such as the Vietnam War, Arab-Israeli conflict, and SALT negotiations.

In a project using human coders, these coding rules are collected into a manual used for training coders; these manuals are often fifty or more pages in length and deal with a variety of contingencies that coders may encounter. The third stage involves training coders so that a news story will be assigned the same codes irrespective of the individual coding it. Coders in event data projects generated in universities are typically graduate students or advanced undergraduates in political science. The training stage is frequently quite time consuming but with sufficient training, most

projects train coders to the point where two coders will assign the same code to a news report in 85% to 90% of the cases (see Burgess and Lawton 1972). In a project dealing with a relatively short period of time – for example the 1990-1991 Gulf crisis – a single researcher may do all of the coding in insure that a single coding standard is used. In machine coding system (see Gerner et al. 1994), a computer program must be provided with the appropriate vocabulary to identify actors and events;

In a machine-coding project, coding rules are implemented in a computer program, usually using extensive dictionaries which identify actors and events and then associate these with specific codes (see Lehnert and Sundheim 1991, Gerner et al 1994). These dictionaries are typically developed by coding a large number of test sentences from the actual data and adding the appropriate vocabulary when the machines makes an error.

When these three tasks have been completed, coding can be done. Generating a large human-coded data set such as WEIS or COPDAB takes a number of years, during which time intercoder reliability must be maintained despite the turnover in the coders. Machine-coding is much faster – a computer program can code hundreds of events per minute – but machine coding is restricted to simple event categories and cannot extract more complicated types of information from a story.

Table 1 shows a sample of the lead sentences of reports on the Reuters newswire that preceded Iraq's invasion of Kuwait in August 1990.[4] Generally each lead corresponds to a single event, though some sentences generate multiple events. For example, the report "July 23, 1990: Iraqi newspapers denounced Kuwait's foreign minister as a U.S. agent Monday" corresponds to an event in the WEIS event coding scheme: the WEIS category 122 is defined as "Denounce; denigrate; abuse." In this event, Iraq is the *source* of the action and Kuwait is the *target*. Together, these generate the event record "900723 IRQ KUW 122" where "900723" is the date of the event, IRQ is a standard code for Iraq, KUW is the code for Kuwait, and 122 is the WEIS category. Table 2 shows the Reuters stories converted to WEIS events.

Table 1
Reuters Chronology of 1990 Iraq-Kuwait Crisis

July 17, 1990: RESURGENT IRAQ SENDS SHOCK WAVES THROUGH GULF ARAB STATES

Iraq President Saddam Hussein launched an attack on Kuwait and the United Arab Emirates (UAE) Tuesday, charging they had conspired with the United States to depress world oil prices through overproduction.

July 23, 1990: IRAQ STEPS UP GULF CRISIS WITH ATTACK ON KUWAITI MINISTER

Iraqi newspapers denounced Kuwait's foreign minister as a U.S. agent Monday, pouring oil on the flames of a Persian Gulf crisis Arab leaders are struggling to stifle with a flurry of diplomacy.

[4]The Reuters reports were downloaded from the NEXIS data service. The full set of reports is considerably more extensive, particularly during the week prior to the invasion.

July 24, 1990: IRAQ WANTS GULF ARAB AID DONORS TO WRITE OFF WAR CREDITS

Debt-burdened Iraq's conflict with Kuwait is partly aimed at persuading Gulf Arab creditors to write off billions of dollars lent during the war with Iran, Gulf-based bankers and diplomats said.

July 24, 1990: IRAQ, TROOPS MASSED IN GULF, DEMANDS $25 OPEC OIL PRICE

Iraq's oil minister hit the OPEC cartel Tuesday with a demand that it must choke supplies until petroleum prices soar to $25 a barrel.

July 25, 1990: IRAQ TELLS EGYPT IT WILL NOT ATTACK KUWAIT

Iraq has given Egypt assurances that it would not attack Kuwait in their current dispute over oil and territory, Arab diplomats said Wednesday.

July 27, 1990: IRAQ WARNS IT WON'T BACK DOWN IN TALKS WITH KUWAIT

Iraq made clear Friday it would take an uncompromising stand at conciliation talks with Kuwait, saying its Persian Gulf neighbor must respond to Baghdad's "legitimate rights" and repair the economic damage it caused.

July 31, 1990: IRAQ INCREASES TROOP LEVELS ON KUWAIT BORDER

Iraq has concentrated nearly 100,000 troops close to the Kuwaiti border, more than triple the number reported a week ago, the Washington Post said in its Tuesday editions.

August 1, 1990: CRISIS TALKS IN JEDDAH BETWEEN IRAQ AND KUWAIT COLLAPSE

Talks on defusing an explosive crisis in the Gulf collapsed Wednesday when Kuwait refused to give in to Iraqi demands for money and territory, a Kuwaiti official said.

August 2, 1990: IRAQ INVADES KUWAIT, OIL PRICES SOAR AS WAR HITS PERSIAN GULF

Iraq invaded Kuwait, ousted its leaders and set up a pro-Baghdad government Thursday in a lightning pre-dawn strike that sent oil prices soaring and world leaders scrambling to douse the flames of war in the strategic Persian Gulf.

Source: Reuters

Table 2
WEIS Coding of 1990 Iraq-Kuwait Crisis

Date	Source	Target	WEIS Code	Type of Action
900717	IRQ	KUW	121	CHARGE
900717	IRQ	UAE	121	CHARGE
900723	IRQ	KUW	122	DENOUNCE
900724	IRQ	ARB	150	DEMAND
900724	IRQ	OPC	150	DEMAND
900725	IRQ	EGY	054	ASSURE
900727	IRQ	KUW	160	WARN
900731	IRQ	KUW	182	MOBILIZATION
900801	KUW	IRQ	112	REFUSE
900802	IRQ	KUW	223	MILITARY FORCE

Event data analysis relies on a large number of events to produce meaningful patterns of interaction. The information provided by any single event is very limited; single events are also affected by erroneous reports and coding errors. However, important events trigger other interactions throughout the system. For example while Iraq's invasion of Kuwait by itself generates only a single event with WEIS code 223 – military force – the invasion triggers an avalanche of additional activity throughout the international system as states and international organizations denounce, approve or comment, so the crisis is very prominent in the event record.

D.3 The History of Event Data in Foreign Policy Analysis

Event data were originally developed by Charles McClelland in the early 1960s as a bridge between the traditional approach of diplomatic history and the new quantitative analysis of international politics advocated in the behavioral approach.[5] McClelland reasoned that history could be decomposed into a sequence of discrete events such as consultations, threats, promises, acts of violence and so forth. Event data formed

[5]There is a fairly substantial paper trail in the development of events data sets, in particular Azar, Brody and McClelland (1972) provide a series of papers coming out of Azar's Michigan State events data conferences in 1969,1970 and 1971; Burgess and Lawton (1972) also covers this period. The early theoretical development of WEIS is thoroughly discussed in a series of papers by McClelland (1967a, 1967b, 1968a, 1968b, 1969,1970); Azar's early development of COPDAB is also fairly well documented (for example Azar and Ben-Dak. 1975, Azar, Cohen, Jukam and McCormick, 1972; Azar and Sloan, 1975).

a link between the then-prevalent general systems theories of international behavior and the textual histories which provided an empirical basis for understanding that behavior. According to McClelland,

> ...International conduct, expressed in terms of event data, is the chief dependent variable of international relations research. ... It is interesting that a starting point is provided as readily by the ordering principle of classical diplomatic history as by the basic concepts of general system analysis. Thus, we may assert that the prime intellectual task in the study of international relations is ... to account for the relations among components of the international system by ... tracing recurring processes within these components, by noting systematically the structure and processes of exchange among the components, and by explaining, finally, the linkages of within-component and between-component phenomena. Obviously the classical definition of diplomatic history is less ponderous and more literary than the general system definition of the task but both ... carry about the same information and involve nearly the same range of choices of inquiry and analysis. (1970,6)

During the 1960s and 1970s, several event data collections were assembled. The COPDAB (Azar 1980, 1982; Azar and Sloan 1975) and WEIS (McClelland 1976) data sets attempt to code all interactions by all states and some non-state actors such as the United Nations and various national liberation movements; the COPDAB and WEIS coding schemes have subsequently been used in a number of other data sets. A variety of domestic and international event data were also collected in the context of more general data sets such as Rudolph Rummel's *Dimensionality of Nations* collection (Rummel 1972), the *World Handbook* (Taylor and Hudson 1972) and various internal conflict data sets collected by Ted R. Gurr (Gurr 1974); these usually focus on a limited set of actions such as uses of force, domestic violence, or changes of government. The *Comparative Research on the Events of Nations* (CREON) data set (Hermann et al 1977), which is specifically designed for the analysis of foreign policy, was also developed during this period.

For a period in the late 1970s and early 1980s, event data were collected by United States governmental agencies such as Department of State, Department of Defense and various intelligence agencies (see Andriole and Hopple 1984; Hopple 1984; Hoople et al 1984; Daly and Andriole 1980; Laurance 1990) and private political consulting firms such as CACI Inc. The Department of State experimented with coding event data for a small set of states in 1971 in its Foreign Relations Indicator Project (FRIP) (see Lanphier 1975). The Pentagon's Defense Advanced Research Project Agency (DARPA) sponsored a large-scale project in the 1970s to develop event data models for crisis forecasting and management, and in the early years of the Reagan administration, a major event data collection and analysis effort was undertaken by the National Security Council staff in the White House.

These efforts apparently had little long-term impact on the formulation of foreign policy, though many of these event data sets are now available in the archives of the Inter-University Consortium for Political and Social Research at the University

of Michigan and are used in research.[6] Laurance (1990) analyzes the reasons for the limited impact of event data on policy: these include the failure to coordinate the event data projects with the analysts and policy-makers who were supposed to use the data, the absence of guidelines on how event data could be used with traditional, non-statistical sources of information, and the absence of user-friendly analytical tools.

Event data collection went into a hiatus in the mid-1980s, though the COPDAB and WEIS data continued to be refined, other data sets such as CREON were used in research, and some new data sets focusing on international crises – notably Russell Leng's Behavioral Correlates of War (BCOW; Leng 1987) and Frank Sherman's SHERFACS (Sherman and Neack 1993) – were developed during this time. Large-scale event data efforts were revived in the early 1990s in the second phase of the National Science Foundation's Data Development in International Relations project (DDIR), directed by Dina Zinnes and Richard Merritt (see Merritt, Muncaster and Zinnes 1993). Rather than simply extending the work of the 1970s, DDIR emphasized the development of new approaches, with particular emphasis on exploiting the computing power available in personal computers and using machine-readable news sources.

D.4 Event Data Sets

Event data sets fall into two general categories: Actor-oriented data sets record all interactions between a set of actors for a specific period of time, for example the Middle East 1949-1969. Episode-oriented sets look at the events involved in a specific historical incident, usually an international crisis or use of force.

D.4.1 Actor-Oriented Data Sets

WEIS

The WEIS coding scheme classifies events into 63 specific categories; these are organized into 22 general categories such as "Consult," "Reward," "Protest," and "Force" (see Table 3). The general categories form a very rough cooperation-conflict continuum. WEIS coding was the *de facto* standard used by the US government-sponsored projects during the 1970s, and consequently a number of the data sets in the ICPSR use the WEIS scheme.

The WEIS data set available at the ICPSR covers only eleven years (1966-1977) and contains only about 90,000 events; the source text is *The New York Times*. Data after 1977 have continued to be coded by McClelland and several of his students – most recently Rodney Tomlinson at the US Naval Academy – but the full series is not available in the public domain at the present time. DDIR has sponsored the

[6]The ICPSR has about two dozen international event data sets; most universities with graduate programs are members of the ICPSR and have access to its archives. Some of the more recent data sets discussed below – for example SAFED, GEDS, CASCON III and SHERFACS – are not presently at the ICPSR; they can usually be obtained from the individual researchers.

development of a machine-coding system for WEIS (Gerner et al, 1994) which could facilitate the generation of WEIS-coded data in the future.

Because most common statistical routines, such as regression analysis, use numerical rather than categorical data, WEIS events are often averaged into numerical scores before being analyzed. Vincent (1979) and Goldstein (1992) provide two such scales that assign numbers on a cooperation-conflict continuum to each WEIS category; Figure 2 was produced using Goldstein's scale. WEIS codes can also be translated into the COPDAB scale, though one cannot translate from COPDAB to WEIS because COPDAB makes fewer distinctions in the type of event.

Table 3
Examples Of WEIS Event Codes

11. REJECT
111 Turn down proposal; reject protest demand; threat
112 Refuse; oppose; refuse to allow
12. ACCUSE
121 Charge, criticize, blame, disapprove
122 Denounce, denigrate, abuse
13. PROTEST
131 Make complaint (not formal)
132 Make formal complaint or protest
17. THREATEN
171 Threat without specific negative sanctions
172 Threat with specific nonmilitary negative sanctions
173 Threat with force specified
174 Ultimatum: threat with negative sanctions and time limit specified
18. DEMONSTRATE
181 Non-military demonstration; walk out on
182 Armed force mobilization, exercise and/or display

Table 4
Examples of COPDAB Event Codes

09 Nation A expressed mild disaffection toward B's policies, objectives, goals, behaviors with A's government objection to these protestations; A's communiqué or note dissatisfied with B's policies in third party

10 Nation A engages in verbal threats, warning, demands and accusations against B; verbal, hostile behavior.

11 Nation A increases its military capabilities and politico-economic resources to counter Nation B's actions or the latter's contemplated actions; A places sanctions on B or hinders B's movement in waterways or on land and attempts to cause economic problems for B.

COPDAB

The COPDAB data set is substantially larger in size and scope than WEIS, with about 350,000 international events for the period 1948-1978. COPDAB uses a number of different news sources rather than depending solely on *The New York Times*; in particular it uses a variety of regional sources to cover events outside of North America and Europe.[7] In contrast to the categories in WEIS, COPDAB uses an ordered coding scheme that goes from 1 to 16 (see Table 4) supplemented by a numerical cooperation-conflict intensity scale developed by Azar and Sloan (1975). COPDAB coding also classifies an event into one of eight types – for example symbolic, political, military, economic or cultural.

Under DDIR sponsorship, a group at the University of Maryland directed by Ted R. Gurr and John Davies is extending the COPDAB data set from 1990 to the present (Davies and McDaniel 1993). Their project, the Global Event Data System (GEDS) is based on the COPDAB framework but uses a much richer data format that preserves much of the original text reporting the event; GEDS also codes a number of internal political actors, particularly ethnic groups.

CREON

The Comparative Research on the Events of Nations data set (Hermann et al 1977; East, Salmore and Hermann 1978) is specifically designed for the study of foreign policy interactions. Its basic event coding scheme is similar to that of WEIS, but CREON in addition codes over 150 variables dealing with the context of the event, related actions, and internal decision-making processes. Unlike WEIS and COPDAB, CREON does not code all interactions during a period of time: instead it covers a random sample of time periods during 1959-1968 and a stratified sample of 36 nation-states which contains a disproportionate number of developed and English-speaking countries. The purpose of CREON is to study the foreign policy *process*, rather than foreign policy output. In practice this means that CREON is better suited than WEIS or COPDAB to studying the linkages between the foreign policy decision-making environment and foreign-policy outputs for specific decisions, but it cannot be used to study policy outputs over a continuous period of time or for countries not in the sample.

Other Actor-Oriented Event Data Sets

While WEIS, COPDAB and CREON are the largest actor-oriented data sets, a variety of smaller sets exist. As noted earlier, the ICPSR has several regionally-specific, WEIS-coded data sets dating from the 1970s, and additional regional data sets are being collected at the present time. The South Africa Event Data set (SAFED; van Wyk and Radloff 1993) is a WEIS-coded collection focusing on southern Africa for the period 1977-1988; it has unusually dense coverage of non-state actors such as guerrilla movements. Ashley (1980) assembled a data set focusing only on the interactions of

[7] Because WEIS and COPDAB are based on different sources, they do not have a high degree of overlap: *International Studies Quarterly* (1983) contains two analyses of this along with a commentary by McClelland.

the superpowers – the USA, USSR and PRC – for 1950-1972; this contains about 15,000 events and is coded with a COPDAB-like scale.

D.4.2 Episode-Oriented Data Sets

BCOW

The Behavioral Correlates of War data set (Leng 1987) codes a sample of 38 major international crises over the period 1816-1975; roughly half of these crises culminated in war and the other half were resolved without war. Most of the crises (31 out of 38) are in the 20th century; about a third (12) are post-WWII; and many of the crises preceding WWI and WWII are included in the sample. BCOW's event codes are an expanded version of the WEIS scheme containing about 100 categories and differentiating more clearly between verbal, economic and military behavior. Leng (1993b) contains an extensive analysis of this data set.

BCOW uses multiple sources of information, including newspaper accounts, diplomatic histories, and chronologies (Leng 1987:1). The number of events in each crisis range from 120 events in the 1889-90 British-Portugal crisis in southern Africa to 2352 events in the 1956 Suez crisis. The ICPSR data set is accompanied by a very extensive coding manual that would allow a researcher to code additional crises in a manner consistent with the original data; it also includes some specialized software that can be used to analyze the data.

Table 5
Examples of BCOW Event Codes

Military Actions (sample from a total of 36 categories)
11212 International Peacekeeping Force
11333 Alert
21143 Change in Combat Force Level
31133 Fortify Occupied Territory
Diplomatic Actions (sample from a total of 35 categories)
12121 Negotiate
12362 Declare Neutrality
12213 Punish of Restrict Foreign Nationals
32151 Grant Independence to Colony
Economic Actions (sample from a total of 20 categories)
13121 Economic Negotiation
23121 Sell or Trade
23231 Pay for Goods or Services
Unofficial Actions (sample from a total of 11 categories)
14251 Proforeign Demonstration
14213 Antiforeign Demonstration
14152 Hostage Taking

CASCON

The Computer-Aided System for the Analysis of Local Conflicts system (CASCON) codes the characteristics of 66 internal and international conflicts during the post-World War II period. The analytical framework is based on a study by Bloomfield and Leiss (1969) and is organized around six predefined conflict phases ranging from the issues leading to the initiation of the dispute to the resolution of the dispute. CASCON codes 540 "factors" for each crisis; some of these describe specific types of events, others describe contextual characteristics of the crisis such as whether the parties to the conflict are dependent on outside aid.

The current version of the data set, CASCON III (Bloomfield and Moulton 1989) is an integrated "decision support system" designed to help decision-makers compare current crises with the historical data on the 66 CASCON crises; the system runs on a personal computer. The CASCON III system contains the conflict data set, a variety of analytical tools that can be used to compare conflicts, and a subsystem for entering new cases into the database. An earlier version on the data set, containing only 52 cases during the 1945-1969 period and without the analytical software, is available from the ICPSR.

SHERFACS

The SHERFACS data set (Sherman and Neack 1993) codes over 700 international disputes and almost 1000 domestic disputes in the 1945-1984 period. It combines several different coding schemes, including COPDAB event codes, the CASCON crisis phase structure, and a variety of conflict management variables originally used in the Butterworth (1976) data set in crisis mediation. SHERFACS is particularly strong on coding nonstate actors such as ethnic groups, transnational actors such as intergovernmental organizations, and non-national actors such as multinational corporations.

An early version of SHERFACS is available from the ICPSR (Alker and Sherman 1982, 1986); the current version is being completed as part of the DDIR project. While SHERFACS is not part of an integrated software system like CASCON, John Mallery and Sigrid Unseld (1992; Mallery forthcoming) have been developing specialized software for analyzing the data and deriving general rules from it. This software is based on artificial intelligence techniques and could be generalized to work with other types of event data.

Other Episode-Oriented Event Data Sets

As noted above, several other data collections available from the ICPSR such as *The World Handbook* contain some limited amounts of event data. Another example is the PRINCE Project data set (Coplin, O'Leary and Shapiro, n.d.). This data set was originally collected in conjunction with a computer simulation project and contains a small set of event data dealing with political issue positions for the period 1 January 1972 to 30 June 1972. Other data sets have been collected for the study of a specific crisis: for example Lebovic (1993) coded events during the period prior to the 1991

Gulf War (2 August 1990 to 16 January 1991) in order to analyze the impact of foreign policy "momentum" in that crisis.

D.5 Applications

Event data have been used in a variety of different studies in foreign policy analysis. This section will briefly discuss five applications that illustrate some of the different analytical techniques employing event data.

D.5.1 Reciprocity in Superpower Interactions

In an extensive analysis reported in their book *Three Way Street*, Joshua Goldstein and John Freeman (1990) combine three event data sets – WEIS, COPDAB and Ashley's superpower data – to create a time series of interactions between the USA, Soviet Union and the People's Republic of China extending from 1948 to 1986. This data is analyzed using a statistical technique called vector autoregression, which assesses the effects of a change in one variable in the system on other variables.

The study is important in two respects. First, the 40-year time series clearly displays the major shifts in the relationships between three major powers, such as the Cold War of the 1950s between the USA and USSR, the *détente* period of the early 1970s and the Reagan-Brezhnev "New Cold War" of the early 1980s. Similarly, the effects of the Cultural Revolution and the Nixon *rapprochement* with China can be seen in the US-Chinese relationship.

Goldstein and Freeman's statistical findings show that most of the interactions between the superpowers were reciprocal – each state received interactions from other superpowers similar to those it projected to them. This pattern of reciprocity had been predicted by a number of theories, and more generally the study of reciprocal behavior has been a major focus of event data research.[8] The study also showed a great deal of inertia in the superpower relationships: the level of cooperation or conflict was maintained about the same level from year to year, changing only slowly.

D.5.2 Political Influence in Arms Transfers

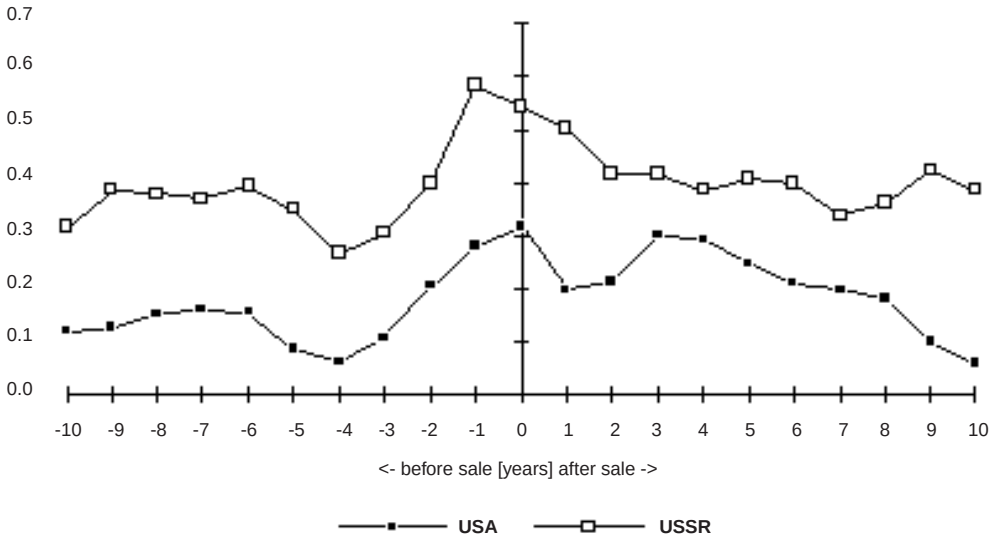
Schrodt (1983) studies the effects of the international sale of weapons on international behavior using event data. One key concern in this arms transfer research is the "arms and influence" relationship: does the supplier of weapons gain political influence over the recipient?

The study uses data from the Stockholm International Peace Research Institute on weapons sales from the USA and USSR to a number of Middle Eastern countries; the COPDAB data set is used to measure cooperative and conflictual behavior between the

[8] See for example Richardson, Kegley and Agnew 1981, Ward 1982, Dixon 1986, Goldstein 1991 and van Wyk and Radloff 1993.

supplier and recipient. The statistical technique was crosscorrelation: the correlation between the level of sales and the cooperative or conflictual behavior at times before and after the arms transfer.

Figure 3
Crosscorrelation of Arms Transfers and International Cooperation from Recipient to Supplier



This technique was successful in demonstrating a number of features of the arms and influence relationship. As expected, there was no significant influence prior to the transfer – except during the one or two years prior to the transfer when it was probably being negotiated – but the data showed statistically significant cooperation lasting for about five years after the transfer.

The contrast between the USA and USSR was even more interesting. As expected, the cooperation of recipients with the Soviet Union was considerably higher than that with the United States: the USSR gained more cooperation, in the short term, from its recipients. However, in the longer term, after about five years, the Soviet Union also had significantly increased *conflict* with its recipients, whereas arms transfers did not significantly change conflictual behavior towards the United States. This results for the Soviet Union had been anticipated in some of the nonstatistical literature (see for example Pierre 1982:81-82), where the tendency of the Soviet Union to alienate its arms recipients was known informally as the "Ugly Russian problem."

D.5.3 Interdependence of International Interactions

Schrodt and Mintz (1988) use the COPDAB data set to study interactions between six Middle Eastern states: Jordan, Syria, Saudi Arabia, Kuwait, Iraq, and Iran during the period 1948 to 1978. The study looked at the probability that an interaction between one pair of nations – for example Syria to Iran – would trigger other interactions, for example Iran to Syria or Saudi Arabia to Iraq.

The study reached a number of conclusions – for example we found that interactions almost always increase, rather than decrease, the probability of other interactions. However, in retrospect our most interesting finding was the prominent role of Kuwait:

> ... when some interaction occurs with Kuwait, this interaction disproportionately sets off other interactions in the system. This initially seems counterintuitive because Kuwait is the least powerful of the states we are studying, though that status may be the reason Kuwait is so important. If this characteristic holds generally, we may find that minor powers are more important in determining interaction interdependence than major powers. (1988:227-228)

This was written in 1984, six years before the 1990-1991 Iraq-Kuwait crisis. The importance of Kuwait was deduced exclusively from the event data itself, rather than from a traditional political analysis.

D.5.4 Decision-Making Units and Foreign Policy

Hermann and Hermann (1989) use the CREON data set to study the effect that the type of foreign policy decision-making unit has on the character of foreign policy. The types of decision-making unit studied are "predominant leader," "single group," and "multiple autonomous actors;" these are described elsewhere in this volume. The nation-states in the CREON data set are coded into these categories according to an explicit set of coding rules; in many cases the category varies due to changes in governments and in some countries (e.g. Switzerland) differs depending on the foreign policy issue. The CREON event data provided the dependent variable, foreign policy behavior, which was coded for affect, commitment and the choice of instruments of statecraft, and the study also controlled for whether the unit was self-contained or could be influenced externally.

The results of the study are clearest on the issue of affect. Hermann and Hermann report:

> The single group decision units engaged in the most extreme behavior of the three types, evidencing the most conflictual behavior. Multiple autonomous actors were the least conflictual, with predominant leaders in between. ... Also as hypothesized, self-contained decision units [a control variable] were significantly more conflictual – that is, more extreme in their behavior – than the externally influenceable units. (1989:380)

In the areas of commitment and choice of instruments, the results are more complex, with interaction effects between the type of decision unit and the control variables. For example, "predominant leaders in self-contained units (the insensitive leaders) use more economic and military instruments of statecraft than those in the externally influenceable units (the sensitive leaders)." (1989: 382).

D.5.5 Influence Strategies in Militarized Interstate Conflicts

Leng (1993a) used the BCOW data to study the relationship between the bargaining strategies employed by states in a dispute and the outcome of the dispute. Starting with the 40 crisis in BCOW and eliminating those crises where no negotiations preceded war, Leng classifies the influence strategies used by 70 parties to the crises into three categories:[9]

Bullying: "the actor employs increasingly severe negative inducements until the other side complies with its demands;"

Reciprocating: "Tit-for-Tat responses to the actions of the other side, along with occasional unilateral cooperative initiatives;"

Trial-and-Error: "the actor simply adjusts its choice of inducements based on the target's response to the preceding influence attempt; ...inducements that produce positive responses are repeated and inducements that produce negative responses are changed."(1993a:5)

These strategies were identified using the events recorded in the BCOW data set.

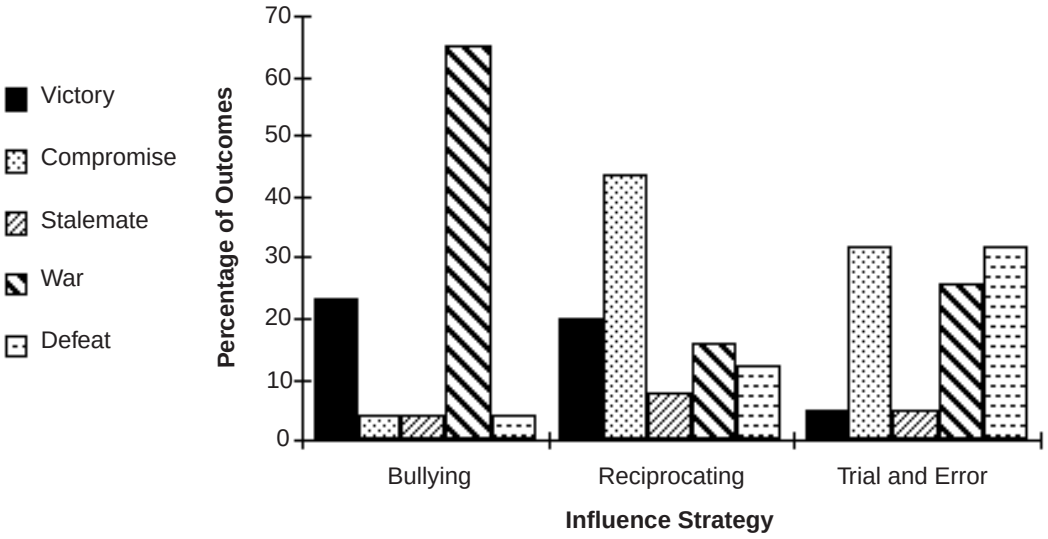
Figure 4 shows the relationship with the strategies used and the crisis outcomes. As Leng observes:

> The comparison between escalating coercive bullying strategies and reciprocating strategies is particularly striking. Escalating bullying strategies leads to war or submission in 69% of the cases, and to a victory or compromise in 27% of the cases; whereas reciprocating influence strategies leads to a victory or compromise in 64% of the cases, and to war or submission in 28% of the cases. When bullying strategies are successful, they do tend to result in diplomatic victories (23%), rather than compromises (4%), but reciprocating strategies also achieve diplomatic victories in 20% of the cases ... along with compromises 44% of the time. (1993a:6)

The trial and error strategy is intermediate between the other two strategies, producing more war and less compromise than the reciprocating strategy but less war and more compromise than the bullying strategy. Leng's results reinforce the theoretical results

[9]Leng also identifies two additional cases where a party used an "appeasement" strategy – both resulted in defeat – and two cases where a party used a "stonewalling" strategyboth resulted in war.

Figure 4
Influence Strategies and Dispute Outcomes



of Axelrod (1984) and a number of other researchers on the value of tit-for-tat strategies in conflict situations.

Problems with Event Data

Event data, like any data used in social research, contain errors due to their source, coding techniques and other factors. The advantages and disadvantages of event data have been extensively studied and discussed; the field is nothing if not introspective.[10] The following is a brief survey of these issues.

D.5.6 Coding Systems

Reflecting the Cold War environment in which they were first developed, the existing coding systems, particularly WEIS and COPDAB, focus primarily on military and diplomatic interactions between nation-states. They provide considerably less detail on economic interactions, newer issues such as refugees, multilateral operations, and environmental regulation, and non-state actors such as international organizations and sub-national groups.[11] This is not a problem if one is primarily interested in diplomatic

[10] See for example Andriole and Hopple 1984, Azar and Ben-Dak 1975, Brody 1972, Burgess and Lawton 1972, Gaddis 1987, *International Studies Quarterly* 1983, Laurance 1990, McGowen et al. 1988, Merritt, Muncaster and Zinnes 1993, Munton 1978, Peterson 1975, Rosenau 1974, and Sigler, Field and Adelman 1972.

[11] Some of the nonstate actors active in the 1970s are coded – for example the United Nations, Irish Republican Army and Palestine Liberation Organization – but the bulk of the interactions in the data sets involve nation-states.

and military interactions between nation-states, but many contemporary studies have a broader focus.

Because of the substantial expense involved with the human coding of events, little experimentation has been done with the implications of alternative coding schemes and the idiosyncrasies of the existing codes have been frozen into place. For example, WEIS has separate codes for a "Warning" and "Threat," though these are often synonymous, and it has only three categories for the use of force, whereas BCOW recognizes about twenty.

Despite its popularity in statistical studies, the conflict-cooperation continuum used by COPDAB and the scaled transformations of WEIS is problematic because there is considerable evidence that conflict and cooperation are *independent* dimensions in international behavior. Nations that have extensive cooperation, for example in trade or alliances, also tend to have greater conflict than nations that are mutually isolated.

Most and Starr (1984) have identified the general empirical problem of "foreign policy substitutability": different actions in foreign policy may have the same general effect. For example, Israel and the Palestine Liberation organization agreed to mutual recognition in 1993 following secret talks mediated in Norway and a public ceremony at the White House, but one could as easily imagine a different set of circumstances where the recognition occurred after secret talks at the United Nations and a public ceremony in Egypt. Depending on the theoretic issue being discussed, these two scenarios might or might not be considered equivalent. The effect of an event data coding scheme is to define a set of equivalent foreign policy actions and assign them identical codes, but the same set of codes may not work equally well for all theoretical questions.

In all likelihood, there will be greater experimentation with new coding systems in the future, particularly as machine-coding systems are developed. The reports of the early event data efforts by researchers such as McClelland and Azar show they had no intention of freezing into place a single standard for event coding; instead they expected that their coding schemes would refined through experience and further theoretical developments. As the cost of coding drops, such experimentation and refinement should be possible.

D.5.7 Source Bias

One of the most widely studied problems in event data collection are the editing and coverage biases introduced by the journalistic sources. One of the earliest systematic studies of this problem was Doran, Pendley and Antunes (1973), who found a dramatically higher level of reported violence in Latin America if they used regional sources rather than international sources. Azar (Azar and Ben Dak, 1975:4) found only a 10% overlap between events reported in *The New York Times* and *Middle East Journal*, with the *MEJ* more likely to report cooperative events. Hoggard (1974) generally finds only 10% to 20% overlap between *The New York Times Index* and regional sources; Gerner et al (1994) report a similar low level of overlap when comparing Reuters with two specialized regional sources.

The interactions of some 180 nation-states are necessarily complex, and it is unlikely that any event data set will capture more than a few percent of all political activities.

However, some events, such as the outbreak of war, are more important in determining international behavior than others, and the likelihood of missing an event is probably inversely proportional to its importance: the more important an event, the more likely it will be reported.

Researchers have taken two different approaches to this problem. Some projects, such as COPDAB, SAFED and BCOW, have used multiple sources to try to capture as many events as possible. This effort is still limited by the time and resources available to the project but as a greater number of machine-readable sources become available, the costs of coding from multiple sources has decreased. Other projects, such as WEIS, CREON and GEDS, have relied on a single source – *The New York Times, Deadline Data,* and Reuters respectively – under the assumption that by maintaining a consistent sources, the *changes* in the patterns of interaction will be more evident.

D.5.8 Additional Variables

All event data sets have in common the use of a basic

<date><source><target><event>

format, but they differ substantially in whether additional information is coded. WEIS codes only the simple format and an optional "arena" code; COPDAB adds an "issue type" code (e.g. diplomatic, military, economic). BCOW and GEDS, in contrast, add dozens of additional variables; SHERFACS and CREON contain hundreds of factors.

Most of this additional information could be categorized as providing "context" for the event. For example, what sub-national decision unit was responsible for the event? What other events were related to it; what other actors were involved? If the event occurs during a crisis, is it part of an escalation or de-escalation? What is the underlying intent of the event, if that can be inferred? In some of the data sets, particularly those dealing with crises, this context, rather than the pattern of discrete events, is the primary focus of the data collection.

The motivation behind adding contextual information to an event record is clear: human decisionmakers perceive events in a very context-rich manner. Human associative memory provides decision-makers with immediate linkages to other events, provides a means of inferring motive and so forth. However, whether one can *systematically* analyze contextual information is an open issue – after all, if one really wants context, one should be reading the original text sources and not bothering with event data in the first place. Most of the existing applications of event data have not used the contextual information and instead have focused on very crude aggregate measures such as moving averages, though this is changing as more sophisticated analytical tools, such as those used with CASCON and SHERFACS, are developed.

D.5.9 Future of event data

While the concept of event data is nearly three decades old, the approach has just begun to enter its second generation. Most of the event data research efforts to date

have been based on concepts and techniques little changed since 1970. However, fundamental changes in the information processing capabilities available to researchers now make possible analytical techniques that were impossible when events data were first developed. Inexpensive personal computers have already passed the speed and mass-storage capacities of university mainframes available in the 1970s, and are rapidly approaching the capacities of supercomputers available in the 1980s. At the same time, many of the sources traditionally used for event data coding have become available in machine-readable form. Consequently, the past may be a poor guide to the future and what was practically impossible a decade ago may be trivial a decade from now. The impact of increased computing power is most clearly reflected in machine coding and new analytical methods.

Events data are very different from the data used in most statistical studies in the social sciences (see Schrodt forthcoming). The conventional statistical repertoire of the social sciences has almost no techniques explicitly adapted to this type of data and, as Achen (1987) points out, there has been virtually no original statistical work to fill these gaps. To date most of the effort in event data analysis has been devoted to carefully constructing and implementing coding schemes rather than systematically exploring what one can do with the data once it has been collected.

McClelland originally envisioned event data as being analyzed as patterns of discrete events.[12] These efforts were unsuccessful: after some years of work with events data focusing on several crises, McClelland concluded,

> It proved relatively easy to discern event patterns and sequences intuitively. We found we could follow the successions of action and response in flow diagram form. Stages of crisis and the linkage of event types to temporary *status quo* situations also were amenable to investigation. We were defeated, however, in the attempt to categorize and measure event sequences. This was an early expectation that was disappointed by the data which showed too few significant sequences to support quantitative or systematic treatment. (1970:33)

With the perspective of two decades of hindsight, the information processing technology and sequence analysis techniques available to McClelland were woefully inadequate. McClelland writes of analyzing hundreds or at most thousands of events; a contemporary events data researcher has available hundreds of thousands of events and would be capable of working with millions.

While many studies of event data use still relatively simple methods, in recent years a variety of more complex techniques have been proposed. Some of these are based on advanced statistical methods such as vector autoregression (Goldstein and Freeman 1990), Poisson regression (King 1989) and event history analysis (Allison 1984). Another set of techniques for event data analysis is found in the computational modelling literature derived from research in artificial intelligence (Hudson 1991; Unseld and Mallery 1992); techniques designed to study molecular sequences (Sankoff and Kruskal 1983) has inspired some other methods; and some computation methods are

[12] Azar, in contrast, saw event data fundamentally in terms of numerical measures; see for example Azar and Ben-Dak (1975). Nonetheless, virtually all event coding schemes other than COPDAB and its derivatives (e.g. GEDS) use categorical coding.

being specifically designed to analyze sequences of social and political events (Heise 1988, Schrodt 1990). Most of these new methods require substantial amounts of computing power and would have been impractical a decade ago, so in the future it may be possible to do considerably more systematic analysis with event data than was possible in the past.

D.5.10 Conclusion

The event data approach demonstrates that it is possible to systematically code a very large number of individual foreign policy interactions and then use that information to test general hypotheses about foreign policy behavior using statistical techniques. These hypotheses may deal with national-level characteristics (Hermann and Hermann 1989); the effectiveness of specific strategies (Schrodt 1983, Leng 1993); patterns of interaction within a subsystem (Schrodt and Mintz 1988, Goldstein and Freeman 1990, van Wyk and Radloff 1993); or patterns in a type of behavior such as a crisis (Sherman and Neack 1993).

The existence of an assortment of event data sets in public archives such as the ICPSR simplifies and systematizes the measurement of many characteristics of interest to analysts of foreign policy behavior. Event data provide a means of controlling, for example, for the effect of the USA-USSR *détente* in studying the foreign policy of the United States or the effects of the Camp David agreements on the foreign policy of Israel. While event data are an imperfect indicator, they are still likely to provide a better measure than alternatives such as assuming the *détente* period coincided with the Nixon administration or that the Camp David agreements had an immediate impact. The behaviors measured by event data may also be at the core of a study: this is particularly true for CREON and the episode-oriented data sets.

The early work in event data analysis was confined to methods that by contemporary standards were slow, laborious and oftentimes of dubious statistical value. The quantum leap in information processing capability in the past decade has clearly opened the way for a distinct second-generation of event data analysis where machine-assisted coding replaces human coding, computer-intensive sequence analysis methods replace descriptive statistics and contingency tables, and analytical software designed to work with specific data sets – currently seen with BCOW, CASCON and SHERFACS – supplements the use of standard statistical packages. The implications of this change for the field of foreign policy analysis are as yet unclear, but they are potentially profound.

Bibliography

[1] Achen, Christopher. 1987. "Statistical Models for Event Data: A Review of Errors-in-Variables Theory." Presented at the DDIR Conference on Event Data, Columbus, OH.

[2] Alker, Hayward R. Jr., and Frank L. Sherman. 1982. "Collective Security Seeking Practices Since 1945." In *Managing International Crises*, ed. Daniel Frei. Beverly Hills: Sage.

[3] Alker, Hayward R. Jr., and Frank L. Sherman. 1986. *International Conflict Episodes, 1945-1979.* (ICPSR 8303) Ann Arbor: Inter-University Consortium for Political and Social Research.

[4] Allison, Paul D. 1984. *Event History Analysis: Regression for Longitudinal Event Data.* Beverly Hills: Sage Publications.

[5] Andriole, Stephen J. and Gerald W. Hopple. 1984. The Rise and Fall of Events Data: From Basic Research to Applied Use in the U.S. Department of Defense. *International Interactions* 11:293-309.

[6] Advanced Research Projects Agency (ARPA). 1993. *Proceedings of the Fifth Message Understanding Conference (MUC-5).* Los Altos,CA: Morgan Kaufmann.

[7] Ashley, Richard K. 1980. *The Political Economy of War and Peace.* London: Frances Pinter.

[8] Azar, Edward E. 1980. "The Conflict and Peace Data Bank (COPDAB) Project." *Journal of Conflict Resolution* 24:143-152.

[9] Azar, Edward E. 1982. *The Codebook of the Conflict and Peace Data Bank (COPDAB).* College Park, MD: Center for International Development, University of Maryland.

[10] Azar, Edward E. and Joseph Ben-Dak (eds.). 1975. *Theory and Practice of Events Research.* New York: Gordon and Breach.

[11] Azar, Edward E., and Thomas Sloan. 1975. *Dimensions of Interaction.* Pittsburgh: University Center for International Studies, University of Pittsburgh

[12] Azar, Edward E., R.D. McLaurin, Thomas Havener, Craig Murphy, Thomas Sloan and Charles H. Wagner. 1977. "A System for Forecasting Strategic Crises: Findings and Speculations About Conflict in the Middle East" *International Interactions* 3:193-222.

[13] Azar, Edward E., Richard A. Brody and Charles A. McClelland. 1972. *International Events Interaction Analysis: Some Research Considerations*. Beverly Hills: Sage Publications.

[14] Azar, Edward E., Stanley H. Cohen, Thomas O. Jukam and James McCormick. 1972. "Making and Measuring the International Event as a Unit of Analysis." In *International Events Interaction Analysis: Some Research Considerations*, ed. Edward E. Azar, Richard A. Brody and Charles A. McClelland. Beverly Hills: Sage Publications.

[15] Bloomfield, Lincoln P., and Allen Moulton. 1989. *CASCON III: Computer-Aided System for Analysis of Local Conflicts*. Cambridge Mass.: MIT Center for International Studies.

[16] Bloomfield, Lincoln P., and Amelia C. Leiss. 1969. *Controlling Small Wars*. New York: Knopf.

[17] Bond, Doug, Bennett, Brad and Vogele, William. 1994. Data development and interaction events analysis using KEDS/PANDA: an interim report. Paper presented at the International Studies Association, Washington.

[18] Brody, Richard A. 1972. "International Events: Problems in Measurement and Analysis." In *International Events Interaction Analysis: Some Research Considerations*, ed. Edward E. Azar, Richard A. Brody and Charles A. McClelland. Beverly Hills: Sage Publications.

[19] Burgess, Philip M. and Raymond W. Lawton. 1972. *Indicators of International Behavior: An Assessment of Events Data Research*. Beverly Hills: Sage Publications.

[20] Butterworth, Robert Lyle. 1976. *Managing Interstate Conflict,1945-74: Data with Synopses*. Pittsburgh: University Center for International Studies.

[21] Coplin, William, Michael O'Leary and Howard Shapiro. n.d. *PRINCE Project: International Transactions, Issue Specific Interactions and Power Data Sets, 1966-1972* (ICPSR 5006). Ann Arbor: Inter-University Consortium for Political and Social Research.

[22] Daly, Judith Ayres, and Stephen J. Andriole. 1980. "The Use of Events/ Interaction Research by the Intelligence Community." *Policy Sciences* 12:215-236.

[23] Davies, John L., and Chad K. McDaniel. 1993. "The Global Event-Data System." In *International Event Data Developments*, ed. Richard L. Merritt, Robert G. Muncaster and Dina A. Zinnes. Ann Arbor: University of Michigan Press.

[24] Dixon, William J. 1986. "Reciprocity in United States-Soviet Relations: Multiple Symmetry or Issue Linkage." *American Journal of Political Science* 30:421-45.

[25] Doran, Charles F., Robert E. Pendley, and George E. Antunes. 1973. "A Test of Cross-National Event Reliability." *International Studies Quarterly* 17: 175-203.

[26] Duffy, Gavan and John C. Mallery. 1986. "RELATUS: An Artificial Intelligence Tool for Natural Language Modeling." International Studies Association, Anaheim.

[27] East, Maurice A., Stephen A. Salmore and Charles F. Hermann, eds. 1978. *Why Nations Act: Theoretical Perspectives for Comparative Foreign Policy Studies*. Beverly Hills: Sage Publications.

[28] Fan, David P. 1988. *Predictions of Public Opinion from the Mass Media*. Westport, CT: Greenwood Press.

[29] Forsyth, Richard and Roy Rada. 1986. *Machine Learning: Applications in Expert Systems and Information Retrieval*. New York: Wiley/Halstead.

[30] Gaddis, John Lewis. 1987. "Expanding the Data Base: Historians, Political Scientists and the Enrichment of Security Studies." *International Security* 12:3-21.

[31] Gerner, Deborah J. 1990. Evolution of a Revolution: The Palestinian Uprising, 1987-1989. International Studies Association, Washington.

[32] Gerner, Deborah J., Philip A. Schrodt, Ronald Francisco and Judith L. Weddle. 1994. "The Analysis of Political Events using Machine Coded Data." *International Studies Quarterly* 38:91-119.

[33] Gerner, Deborah J., Philip A. Schrodt, Ronald Francisco, Judith Weddle and Julia Pitner. 1992. "Machine Coding of International Events using International and Regional Sources." Paper presented at the International Studies Association, Atlanta, April 1992.

[34] Goldstein, Joshua S. 1991. "Reciprocity in Superpower Relations: An Empirical Analysis." *International Studies Quarterly* 35:195-209.

[35] Goldstein, Joshua S. 1992. "A Conflict-Cooperation Scale for WEIS Events Data." *Journal of Conflict Resolution* 36: 369-385.

[36] Goldstein, Joshua S., and John R. Freeman. 1990. *Three-Way Street: Strategic Reciprocity in World Politics*. Chicago: University of Chicago Press.

[37] Gurr, Ted Robert. 1974. *Civil Strife Events, 1955-1970* (ICPSR 7531). Ann Arbor: Inter-University Consortium for Political and Social Research.

[38] Heise, David. 1988. "Modeling Event Structures." *Journal of Mathematical Sociology* 13:138-168.

[39] Hermann, Charles F. 1987. "Observations from the First Generation of International Event Data Research." Presented at the DDIR Conference on Events Data, Columbus, OH.

[40] Hermann, Charles, Maurice A. East, Margaret G. Hermann, Barbara G. Salmore, and Stephen A. Salmore. 1973. CREON: *A Foreign Events Data Set*. Beverly Hills: Sage Publications.

[41] Hermann, Margaret G., and Charles F. Hermann. 1989. "Who Makes Foreign Policy Decisions and How: An Empirical Inquiry." *International Studies Quarterly* 33:361-387.

[42] Hoggard, Gary. 1974. "Differential Source Coverage in Foreign Policy Analysis." In *Comparing Foreign Polcies*, ed. James Rosenau. New York: Wiley.

[43] Hoople, Gerald W., Stephen J. Andriole and Amos Freedy, eds. 1984. *National Security Crisis Forecasting and Management*. Boulder: Westview Press.

[44] Hopple, Gerald W. 1984. Computer-Based Early Warning: A Staircase Display Option for International Affairs Crisis Projection and Monitoring. pp. 47-84 in Gerald W. Hopple, Stephen J. Andriole and Amos Freedy. *National Security Crisis Forecasting and Management*. Boulder: Westview Press.

[45] Howell, Llewellyn D, Sheree Groves, Erin Morita and Joyce Mullen. 1986. Changing Priorities: Putting the Data back into Events Data Analysis. International Studies Association, Anaheim.

[46] Hudson, Valerie M. ed. 1991. *Artificial Intelligence and International Politics*. Boulder: Westview Press.

[47] International Studies Quarterly. 1983. Symposium: Events Data Collections. *International Studies Quarterly* 27.

[48] Jervis, Robert (1976) *Perception and Misperception in International Politics*. Princeton: Princeton University Press.

[49] King, Gary. 1989. "Event Count Models for International Relations: Generalizations and Applications." *International Studies Quarterly* 33:123-148.

[50] Krippendorff, Klaus. 1980. *Content Analysis*. Beverly Hills: Sage.

[51] Krippendorff, Klaus. 1980. *Content Analysis: An Introduction to its Methodology*. Beverly Hills: Sage Publications.

[52] Lanphier, Vernard A. 1975. "Foreign Relations Indicator Project" In *Theory and Practice of Events Research*, ed. Edward E. Azar and Joseph Ben-Dak. New York: Gordon and Breach.

[53] Laurence, Edward J. 1990. Events Data and Policy Analysis. Policy Sciences 23:111-132.

[54] Lebovic, James H. 1993. "Before the Storm: Momentum and the Onset of the Gulf War". Presented at the annual meeting of the International Studies Association, Acapulco.

[55] Lebow, Richard Ned (1981) *Between Peace and War: The Nature of International Crises*. Baltimore: Johns Hopkins University Press.

[56] Lehnert, Wendy and Beth Sundheim. 1991. "A Performance Evaluation of Text Analysis." *AI Magazine* 12:81-94.

[57] Leng, Russell J. 1987. *Behavioral Correlates of War*, 1816-1975. (ICPSR 8606). Ann Arbor: Inter-University Consortium for Political and Social Research.

[58] Leng, Russell J. 1993a. "Reciprocating Influence Strategies in Interstate Crisis Bargaining." *Journal of Conflict Resolution* 37:3-41.

[59] Leng, Russell J. 1993b. *Interstate Crisis Behavior, 1816-1980*. New York: Cambridge University Press.

[60] Lovins, J. B. 1968. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*. 11:1-2, 11-31.

[61] Mallery, John C. forthcoming. "Beyond Correlation: Bringing Artificial Intelligence to Events Data." *International Interactions*.

[62] McClelland, Charles A. 1967a. "Event-Interaction Analysis in the Setting of Quantitative International Relations Research." mimeo, University of Southern California.

[63] McClelland, Charles A. 1967b. "World-Event-Interaction-Survey: A Research Project on the Theory and Measurement of International Interaction and Transaction." mimeo, University of Southern California.

[64] McClelland, Charles A. 1968a. "International Interaction Analysis: Basic Research and some Practical Uses." mimeo, University of Southern California.

[65] McClelland, Charles A. 1968b. "Access to Berlin: The Quantity and Variety of Events, 1948-1963." In *Qantitative International Politics*, ed. J. David Singer. New York: Free Press

[66] McClelland, Charles A. 1969. "International Interaction Analysis in the Predictive Mode." mimeo, University of Southern California.

[67] McClelland, Charles A. 1970. "Some Effects on Theory from the International Event Analysis Movement." mimeo, University of Southern California.

[68] McClelland, Charles A. 1976. *World Event/Interaction Survey Codebook.* (ICPSR 5211). Ann Arbor: Inter-University Consortium for Political and Social Research.

[69] McClelland, Charles A. 1983. Let the User Beware. *International Studies Quarterly* 27,2:169-177

[70] McGowan, Patrick, Harvey Starr, Gretchen Hower, Richard L. Merrit and Dina A. Zinnes. 1988. International Data as a National Resource. *International Interactions* 14,2:101-113.

[71] McGowan, Patrick, Harvey Starr, Gretchen Hower, Richard L. Merritt and Dina A. Zinnes. 1988. "International Data as a National Resource." *International Interactions* 14:101-113.

[72] Merritt, Richard L. 1987. Event Data: A Synoptic View. Paper presented at the First DDIR Conference on Event Data, Columbus, OH, May 1987.

[73] Merritt, Richard L., Robert G. Muncaster, and Dina A. Zinnes, eds. 1994. *Management of International Events: DDIR Phase II.* (Ann Arbor: University of Michigan Press.

[74] Most, Benjamin A., and Harvey Starr. 1984. "International relations theory, foreign policy substitutability and 'nice' laws." *World Politics* 36:383-406

[75] Munton. D. 1978. *Measuring International Behavior: Public Sources, Events and Validity.* Dalhousie University: Centre for Foreign Policy Studies.

[76] Peterson, Sophia. 1975. "Research on research: Events data studies, 1961-1972." In *Sage International Yearbook on Foreign Policy Studies*, ed. Patrick J. McGowan. Beverly Hills:Sage.

[77] Pierce, John R. 1980. *An Introduction to Information Theory.* New York: Dover.

[78] Pierre, Andrew J. 1982. *The Global Politics of Arms Sales.* Princeton: Princeton University Press.

[79] Pinker, Steven. 1994. *The Language Instinct.* New York: W. Morrow and Co.

[80] Richardson, Neil R., Charles W. Kegley and Ann C. Agnew. 1981. "Symmetry and Reciprocity as Characteristics of Dyadic Foreign Policy Behavior." *Social Science Quarterly* 62:128-138.

[81] Rosenau, James. ed. 1974 *Comparing Foreign Policies.* New York: Wiley.

[82] Rummel, Rudolph J. 1972. *The Dimensions of Nations.* Beverly Hills: Sage Publications.

[83] Salton, Gerald. 1989. *Automatic Text Processing.* Reading, Mass: Addison-Wesley.

[84] Sankoff, David, and Joseph B. Kruskal, eds. 1983. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. New York: Addison-Wesley.

[85] Schrodt, Philip A. 1983. "The Effects of Arms Transfers on Supplier-Recipient Behavior." Presented at the International Studies Association, Mexico City.

[86] Schrodt, Philip A. 1990. "Parallel Event Sequences in International Crises." *Political Behavior* 12:97-123.

[87] Schrodt, Philip A. 1991. "Pattern Recognition in International Event Sequences: A Machine Learning Approach." In *Artificial Intelligence and International Politics*, ed. Valerie M. Hudson. Boulder: Westview Press.

[88] Schrodt, Philip A. 1993. The Machine Coding of Events Data. in Richard L. Merritt, Robert G. Muncaster and Dina A. Zinnes, eds. *Management of International Events: DDIR Phase II*. Ann Arbor: University of Michigan Press. 1993.

[89] Schrodt, Philip A. 1994. "Statistical Characteristics of Events Data." *International Interactions* 20,1-2: 35-53

[90] Schrodt, Philip A. and Christopher Donald. 1990. Machine Coding of Event Data. International Studies Association, Washington.

[91] Schrodt, Philip A. and David Leibsohn. 1985. An Algorithm for the Classification of WEIS Event Code from WEIS Textual Descriptions. International Studies Association, Washington

[92] Schrodt, Philip A. and Deborah J. Gerner. 1994. "Validity Assessment of a Machine-Coded Event Data Set for the Middle East, 1982-1992." *American Journal of Political Science* 38:825-854.

[93] Schrodt, Philip A., and Alex Mintz. 1988. "A Conditional Probability Analysis of Regional Interactions in the Middle East." *American Journal of Political Science* 32:217-230

[94] Sherman, Frank L., and Laura Neack. 1993. "Imagining the Possibilities: The Possibilities of Isolating the Genome of International Conflict From the SHERFACS Dataset." In *International Event Data Developments*, ed. Richard L. Merritt, Robert G. Muncaster and Dina A. Zinnes. Ann Arbor: University of Michigan Press.

[95] Sigler, John H., John O. Field and Murray L. Adelman. 1972. *Applications of Events Data Analysis: Cases, Issues and Programs in International Interaction*. Beverly Hills: Sage.

[96] Stone, P.J., D.C. Dunphy, M.S. Smith and D.M. Ogilvie. 1966. *The General Inquirer: A Computer Approach to Content Analysis*. Cambridge: MIT Press.

[97] Taylor, Charles L. and Michael C. Hudson. 1972. *World Handbook of Political and Social Indicators*, 2nd ed. New Haven, CT: Yale University Press.

[98] Tomita, Masaru. 1986. *Efficient Parsing for Natural Language*. Boston: Kluwer Academic Publishers.

[99] Unseld, Sigrid D., and John C. Mallery. 1992. "Interaction Detection in Complex Datamodels" (AI Memo No. 1298). Cambridge: MIT Artificial Intelligence Laboratory.

[100] van Rijsberger, C.J. 1979. *Information Retrieval* (2nd edition). London: Butterworths.

[101] van Wyk, Koos and Sarah Radloff. 1993. "Symmetry and Reciprocity in South Africa's Foreign Policy." *Journal of Conflict Resolution* 37:382-396.

[102] Vertzberger, Yaacov Y. I. (1990) *The World in Their Minds: Information Processing, Cognition and Perception in Foreign Policy Decision Making*. Stanford: Stanford University Press.

[103] Vincent, Jack E. 1979. *Project Theory: Interpretations and Policy Relevance*. Washington: University Press of America.

[104] Ward, Michael Don. 1982. "Cooperation and Conflict in Foreign Policy Behavior." *International Studies Quarterly* 26:87-126.

[105] Weber, Robert Philip. 1990. *Basic Content Analysis*. Newbury Park, CA: Sage Publications.