SMASER

TRIPmanager User’s Guide

TRIPsystem
Product Documentation

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether "Software") supplied by Smaser AG (Smaser) are exclusively owned by Smaser.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of Smaser. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by Smaser to the extent specified in his contract with Smaser.

The free-to-use non-commercial version doesn't require a prior written agreement with Smaser but such customers, organizations and/or third parties agree by using the software and / or solution of Smaser to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely uninfringed and protected.

Table of Contents

About This Guide

Status of this Guide

This guide is updated to TRIP v7 status. While every effort has been made to ensure this guide is as up-to-date as possible, please refer to the TRIPsystem release notes and the TRIPmmc.chm help file for any 'hot off the press' information about TRIPmanager.

Scope and Assumptions

This guide describes the use of TRIPsystem version 7.0.0 or later via the TRIPmanager plug-in for Microsoft Management Console (mmc), which encompasses the creation and maintenance of databases and the management of user access to the system and its databases.

It is assumed that anyone using the TRIPmanager plug-in is already familiar with the Windows operating system in general and with the mmc in particular.

Further guidance can be found in the contextual help systems provided with the mmc and also with the TRIPmanager plug-in. In addition, help about TRIPmanager can also be reached directly via the TRIPmmc.chm help file in the TRIPmanager installation directory.

Conventions Used in this Guide

Certain symbols and conventions are used throughout this manual to indicate words or phrases with special meanings. A word might indicate the name of a key on the keyboard (<**Tab**>), an option in the menus (**CCL Search**), one of TRIP's command words (**DEfine** or **DE**), the name of a database (**Alice**) or a word being searched for (**wonderland**). The conventions and styles used are summarized below:

italic	used to indicate variables such as *fieldtype* or *databasename*, and to emphasize important terms and concepts
bold	used to indicate anything that TRIP recognizes or can interpret and act upon, such as the things mentioned above (**<Tab>**, **CCL Search**, **DEfine**, **Alice**, and **wonderland**)
lower case	used for terms and variables where variables are also italic
Upper Case	used for proper names such as the database **Alice**
`Courier fonts`	examples containing specific text which you are to type in
< >	chevrons—used to indicate key(s) on the keyboard such as <**Tab**> or <**Enter**>
↵	left arrow used after commands, meaning 'press either <**Enter**>or <**Return**>'
<**Next**>	the <**Page Down**> or <**Next Page**> key
<**Prev**>	the <**Page Up**> or <**Previous Page**> key
" "	messages provided by TRIP

Note:

In this and other manuals of the TRIP product family, the terms 'order' and 'command' are synonymous to each other, as are the terms 'modifier' and 'function'; e.g. in 'DEfine Fuzz', the modifier/function is 'FUZz', while the CCL order/command is 'DEFine'.

Structure of this Guide

This manual is divided into six main parts:

- ***Walk Through Tutorial Sessions***: Chapters One through Three contain three command-driven search tutorials, which cover the use of the main search commands. We encourage you to work through each tutorial in sequence (and, if possible, each succeeding chapter), and to try the examples given using the demonstration databases provided.
- ***Advanced Searching***: Chapters Four through Eight cover more advanced search techniques including, best match searching, classification searching, non-Boolean searching and indirect searching using the command line.
- ***Data Entry Forms***: Chapter Nine.
- ***User Administration and Procedures***: Chapters Ten and Eleven.
- ***Macros***: Chapter Twelve.
- ***Appendices***: CLI Switches.

The chapters are divided into short sections, each introducing a single concept and giving examples where appropriate. These can be used either for reference or as tutorials, repeating the examples given in the demonstration databases **Alice**, **Carroll**, **Corr** and **Thesali**.

The TRIP Documentation Library

Other members of the TRIP documentation library include, but are not limited to, the *Installation Guides*, *Release Notes*, *Release Histories*, *TRIPmanager Administration Guide*, *TRIPclassic Administration guide*, *TRIPmanager User Guide*, *CCL Command Reference*, the *TRIPtoolkit.chm* help file and the notes and help files included with TRIP add-ons such as the programming APIs.

If you are not already familiar with searching and CCL (TRIP's Common Command Language), we recommend that you first read the Introduction, followed by the self-tutorials in chapters one through three, placing special emphasis on the basic commands Find, Display, Show, Print, DEfine, List, BASe, DELete, STatus, Run, SAve and Help.

Note:

Additional information on the above CCL commands (and others) is available in the CCL Command Reference.

What TRIP Recognises

TRIP possesses its own vocabulary, which consists of any word of the Common Command Language (CCL), i.e. all operators, modifiers, qualifiers and keywords. It also acknowledges all real names for fields, records, views, databases etc. in the current TRIP environment.

Variables are occasionally used both in text and examples, which are single symbols or words which may be replaced by a constant (i.e. a proper name, term or value).

Fieldname, *fieldtype*, *viewname*, *recordno*, and *databasename* are examples of variables.

Be sure to distinguish the difference between a field itself, a field's name (such as **person** in the database **Alice**), and the variable *fieldname*, which may be substituted by any real field's name.

Introduction

The Content of this Guide

This guide presents basic skills and concepts to get you started in TRIP for UNIX and Windows, using the TRIPmanager plug-in for the Microsoft Management Console.

In TRIP, *search orders* are the strings of symbols and characters used to locate information in a database. These can be typed into a command entry box on the screen.

Navigation within TRIPmanager

After connecting to a TRIPsystem server, an mmc window will be similar in appearance to the screenshot below, which shows a connection to the TRIP server 'My Computer':

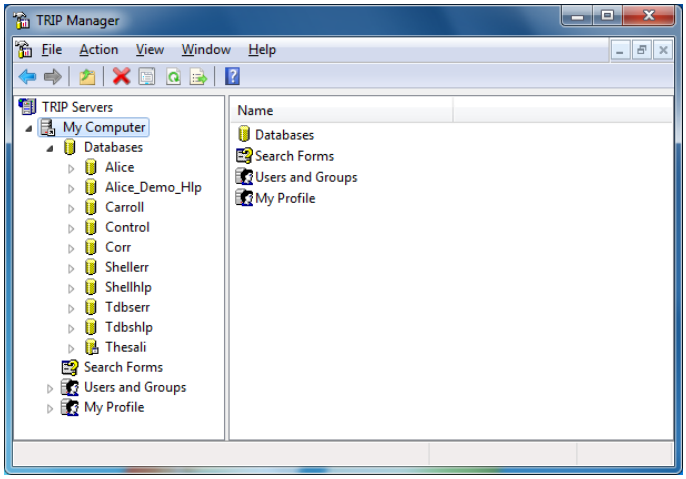

Figure 0–1 The mmc showing a TRIP server connection

Each TRIP server connection will show four main icons representing sub-groupings of items relating to the server being managed. These icons are:

- Databases

  All databases accessible to use administrator accessing the server in question

- Search forms

  All search forms accessible to the administrator accessing the server in question

- Users and Groups

  All search forms accessible to use administrator accessing the server in question, assuming the username is granted user manager rights

- My Profile

  The profile belonging to the currently logged on user and will be covered in detail later in this guide.

This guide makes frequent reference to the 'action' menu. Regular users of Windows will, no doubt, be aware that menu items within the mmc are contextual, as in most Windows applications. This is also the case with TRIPmanager, hence it is not absolutely necessary to use the physical 'Action' menu as it is also possible to right click on an item to reveal its contextual menu. Therefore, where clarity is paramount, screenshots may actually show contextual menus rather than the 'Action' menu.

The TRIPmanager CCL Command Window

Throughout the search sessions in this guide, we will be using the demonstration database **Corr** as an example. This very small database contains **Corr**espondence to and from a software company in the form of letters and telexes. Each piece of correspondence constitutes a record and contains the following information: the date, category and text of each message, and the name, company name, address and country of each sender and recipient, as well as any modifications made to the record.

To obtain a CCL window with a the database Corr pre-opened, click on the database to highlight it, then select 'CCL Query…' from the action menu of the mmc:

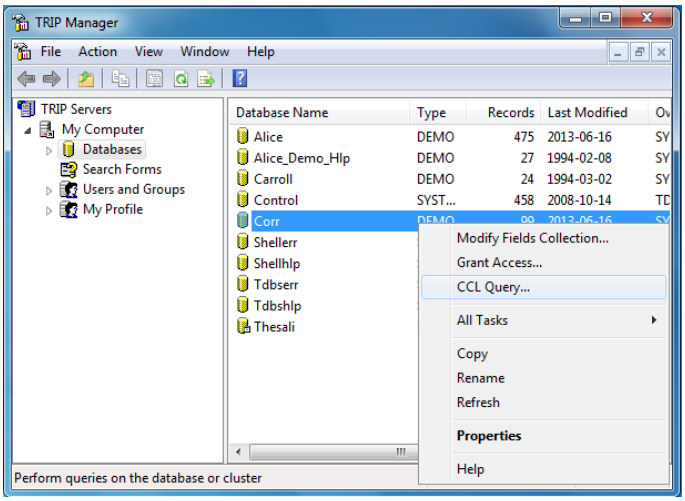

Figure 0–2 Opening a CCL command window for CORR

a window similar to Figure 0–2 will then appear.

Note:

In order to obtain a CCL command window with no pre-opened database, simply ensure that no database has been highlighted before selecting 'CCL Query…' from the action menu.

Figure 0–2 highlights three of these areas; at the top is the Command History pane. Below that are the 'Previous commands' pane and the Command entry box.

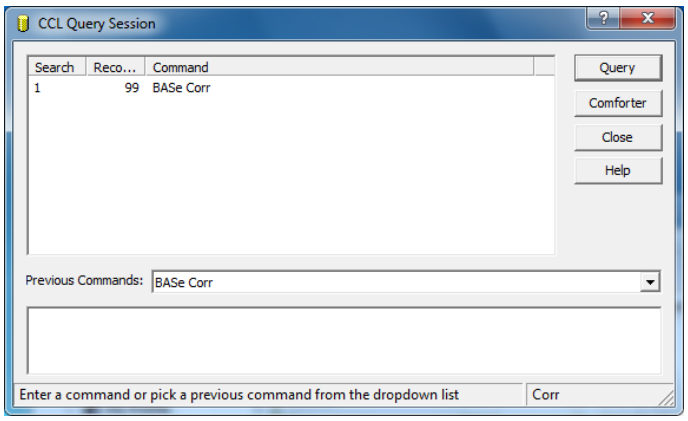

Figure 0–3 A CCL command window in TRIPmanager

Note:

Clicking on the 'Query' button will submit any CCL command entered into the entry box at the bottom of the window: The results or responses from the system are shown in various separate pop-up windows.

Command entry box searching

Command entry box searching uses **CCL** or **Common Command Language** to tell the system what you want to look for, and what you want to do with the results of your searches. CCL searches are created using a handful of simple commands and the words or phrases (terms) that you want to find.

To make searching for such a wide variety of information easier, each record is divided into fields, with the name of the sender in one field, the date in another, and so on. Each field has a name that you can refer to when searching.

Different types of data are stored in different types of fields. In TRIP there are six searchable data types, **TExt**, **PHrase**, **NUmber**, **INteger**, **DAte** and **TIme**, and one non-searchable data type, **STring**, which is intended for storage of binary data such as photographs or vocal annotations.

Four of these data types are used in database **Corr,** which is organised in this fashion:

Field Name	Data Type	Contains	
rname	PHrase	Recipient:	name
rcomp	"	"	company
raddr	"	"	address
rcountry	"	"	country
sname	"	Sender:	name
scomp	"	"	company
saddr	"	"	address
scountry	"	"	country
day	DAte	Date of the message	
cat	PHrase	Type of documentation	
content	TExt	Text of the message	
modified	PHrase	Modified by (TRIP username)	
moddate	DAte	Modification:	date
modtime	TIme	"	time
modnote	PHrase	Modifier's note	

Part 1:

Self-Tutorial Sessions

Chapter 1:
Tutorial One

This first tutorial session assumes that you already have a CCL command window with the Corr database opened and covers the performing of a basic search, displaying and printing the results and ending a search session.

Searching

In the first search example we will be examining only those fields in **Corr** which contain text (data types **TExt** and **PHrase**) and ignoring those that do not (data types **DAte** and **TIme**).

A TRIP search order or request must begin with the command **Find**, which can be shortened to **F**. Start by looking for a word that is almost certain to occur in **Corr**, say, any reference to computer systems (remember, it contains correspondence to and from a software company). Type the following order in the box where the cursor is, and press ↵:

```
f system ↵
```

Your cursor is again returned to the Command window (as it will be after every **Find** command you issue) and in the Search History box the following information appears (Figure 1–7):

```
Search  Records   Command
1       99        BASe Corr
2       66        Find system
```

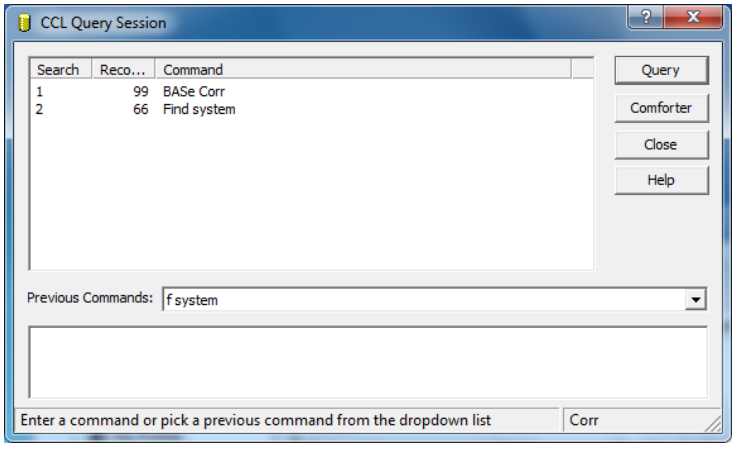

Figure 1–4 The first search screen

This tells you that a second search result has been formed (S=2), and that it consists of sixty-six records (<66>) which all contain the word ‘system’. These are referred to as the *hit records*, and sixty-five is the *hit record count*.

Suppose you wish to narrow down the search by specifying that you are only looking for correspondence regarding computer systems and university applications. Type the order:

```
f system and universities ↵
```

A new line is added to the Search History (Figure 1–8):

```
Search  Records    Command
3       3          Find system AND universities
```

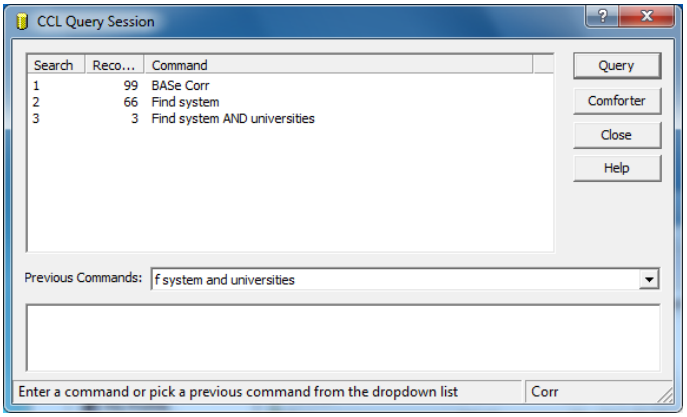

Figure 1–5 The second search screen

This tells you that a third search result has been formed (S=3) and that it consists of only three records (<3>) that contain both 'system' and 'universities'. The word **AND** is a type of symbol known as a *logical operator* that combines one search condition with another in a search order.

It is possible that by using the term 'universities' (plural) in the last search order we may have missed some records which contain the word 'university' (singular). However, instead of writing both forms of the word in the order, we can use the truncation symbol # following the word stem, e.g. 'universit#', to find all relevant records.

A search condition need not be a word, as in the above order, but can be a reference to an earlier search result as well. Try typing the order:

```
f s=2 and universit# ↵
```

This search uses the result of our previous search (number two, 'Find system'), to locate all correspondence containing both the word 'system' and words that begin with 'universit', whatever their suffix. This search finds eighteen records.

The Search History now shows the following (Figure 1–9):

```
Search  Records    Command
4       19         Find S=2 AND universit#
```

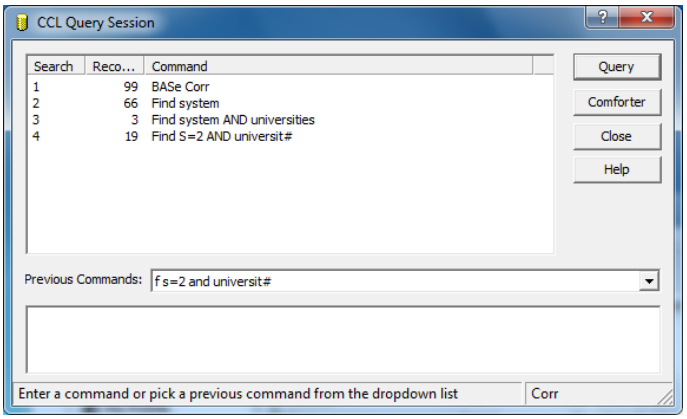

Figure 1–6 The third search screen

As you can see, TRIP appends each search result to the previous one. The Search History window now contains these notations:

```
Search  Records    Command
1       99         BASe Corr
2       66         Find system
3       3          Find system AND universities
4       19         Find S=2 AND universit#
```

Showing Search Results

To view the records found by the last search order, type the order **Show** (this can be shortened to **S**):

s ↵

TRIP opens a **S**how window and displays the first record of your search result as a standard TRIPclassic screen output (Figure 1–10):

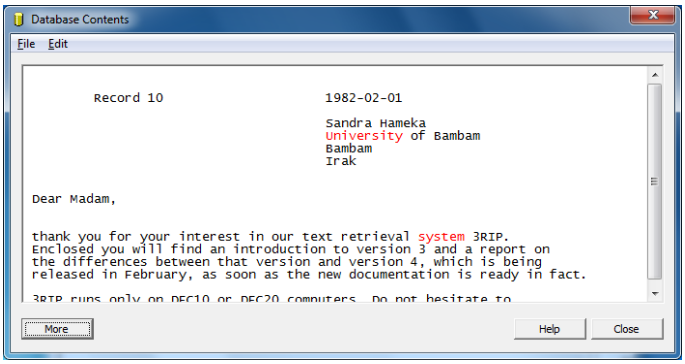

Figure 1–7 The Show window

The ‘More’ button can be used to add the next page of results to the **s**how window. The horizontal and vertical scroll bars can then be used to view the contents of the window.

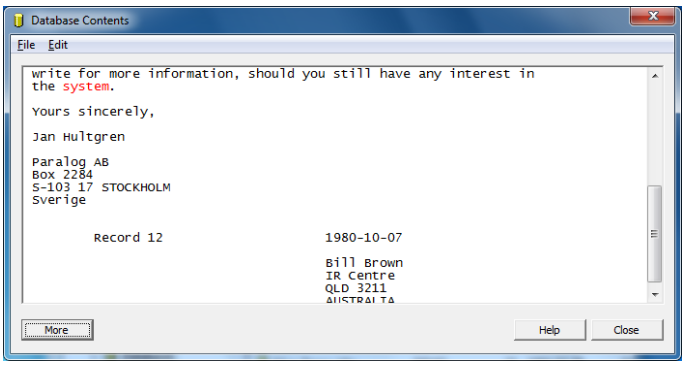

Figure 1–8 More of the Show Window

Clicking on the ‘Close’ button will close the window.

If you are looking at your search results in the Show window and wish to see the list of Find commands you have used in the History window, it will be necessary to click on the Command window, or use the key combination, <Alt><Tab>, to return to it.

There is always a default report for a database—that is, the one used if no other is specified—and will be either the system format (called Dump) or a format designated by the Application Manager.

The records in these examples are shown in **Corr**'s default report, which is named '1'. This particular format shows all of the information contained in each record, including its record number or unique database identifier.

Note:

> *You will not receive notification from TRIP that you have performed a **Show** so it will not be added to the 'Previous Commands:' drop-down selection box, as were the **BASe** and **Find** commands, but selecting a command from this box will automatically load it into the command entry box ready for editing and/or executing.*

If you wish to see the records of a search result other than the current one (the last search number shown in History), add the number of that result to your **Show** order, like this:

```
s s=3 ↵
```

Try this now.

Printing Search Results

Suppose you wanted a hard copy (printed or paper) of the records you found with (S=3) above.

If you type the order **Print** in the Command box and press ↵, you will receive a pop-up message: "Print request no 1 stored." TRIP puts the records of the last search result into a queue, from which it will be output when you end your TRIP session. Other variations of the **Print** command are available and will be discussed later.

Ending the Search Session

To end a search session and return to the Primary Option Menu, click on the 'Close' button. This ends the search session, closes the database and returns you to the area from which your TRIP CCL search session was initiated.

Chapter 2: Tutorial Two

This session gives examples of more refined searches, using date, time and number information as well as text within the database records. It shows how to make use of the database vocabulary, and how to change reports for **Show** and **Print** orders. The tutorial concludes with some information about help and other messages from TRIP.

Log on to TRIP and choose **Search**, then **CCL Search** as in the first session. Open the database **Corr** by typing the order **BASe Corr** (or **bas corr** for short) on the command line. When TRIP shows the order

```
Search  Records   Command
1       99        BASe Corr
```

in the Search History box, you can begin searching.

You may find it easier during the coming exercises to use each search statement as a template for the next. To do this, use the drop-down 'Previous Command:' selection box. Edit the retrieved command as you normally would.

In the first session **AND** (the **AND operator**) was used to combine or add search conditions, or specify conditions where any records found must satisfy one condition **AND** the other. To specify alternative search conditions, or find records which must satisfy one condition **OR** the other (or both), use the **OR operator**.

For example, the **Find** order

```
f price or lease ↵
```

locates nineteen records that contain 'price', 'lease' or both 'price' and 'lease', as seen in the History window:

```
Search  Records   Command
1       99        BASe Corr
2       19        Find price OR lease
```

For the remainder of this chapter, each CCL example will have the corresponding Search History notation printed directly below the command illustrated.

You can search for one term OR the other, but not both, with the exclusive OR operator **XOR**:

```
f price xor lease ↵
Search  Records   Command
1       99        BASe Corr
…
3       17        Find price XOR lease
```

which looks for all records which contain either the word 'price' OR the word 'lease', but not both.

You can also request that a search term be excluded from records that contain another search term. This order,

```
f price not lease ↵
Search  Records   Command
…
```

```
4        14          Find price NOT lease
```

finds fourteen records that contain ‘price’, but NOT ‘lease’.

The effect of **AND**, **OR**, **XOR** and **NOT** is usually illustrated by Venn Diagrams, as shown below:

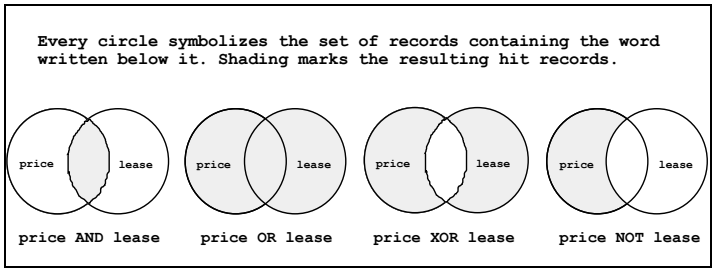

Figure 2–9 The Effect of AND, OR, XOR and NOT

If you have not already done so, type the four orders given previously, as the results are used in examples later in this section.

Searching for Numbers, Dates and Times

As mentioned earlier, a record is made up of fields which have names and that contain different types of data. Normally the data types TExt and PHrase are the default—that is, are automatically searched—if no field names are specified in a search order (all the search orders of the first session found occurrences in the TExt and PHrase fields of the records). In addition to the TExt and PHrase fields, the **Corr** database has two fields of type **DAte** called **day** and **moddate**, and one of type **TIme** called **modtime**.

The field we are interested in, **day**, contains the date each piece of correspondence was written. The system default format for dates is country-specific; however you may override this by choosing any one of seventeen possible date formats. Also, while doing these exercises you may receive the message “*Date* is no legal date” beneath the Command window, where *date* is the date format given in the sample command. Should this happen, refer to the section called ‘User Profile’ in Chapter Ten of this guide for information on finding your default date form, and if you wish, changing it.

Suppose you are interested only in those letters located by search result number two that are dated later than June 1984. Type the **Find** order:

```
f s=2 and day>1984-6-30 ↵
Search  Records   Command
…
5       12        Find S=2 AND day > 1984-06-30
```

The following **Find** order would give the same result:

```
f s=2 and day>=1984-7 ↵
Search  Records   Command
…
6       12        Find S=2 AND day >= 1984-07
```

Notice that TRIP automatically capitalizes logical search operators (**AND**, **OR**, **XOR**, **NOT**), expands dates (month 7 becomes 07) and inserts spaces in your commands where appropriate for readability.

These examples show that a full date is not always required. A mathematical search operator (equality sign [=], with or without a sign for greater than [>] or less than [<]) is placed between the field name and the value. Instead of greater than and less than, **FRom** (short form: **FR**) and **TO** may also be used to find values which are equal to, as well as greater or less than, the values specified.

Using **FRom**, the above order then becomes:

```
f s=2 and day=fr 1984-7 ↵
Search  Records   Command
…
7       12        Find S=2 AND day=FRom 1984-07
```

If you want only those letters written in July 1984 or later than September 1984, give the order:

```
f s=2 and day=1984-7 fr 1984-10 ↵
Search  Records   Command
…
8       9         Find S=2 AND day=1984-07 FRom
1984-10
```

Here you see that it is not necessary to be specific about days of the month in the query. TRIP calculates the number of days in the month specified in the search request (including allowances for leap years) and finds all correspondence for that month (here, July 1984 *or* on or after October 1984), saving many keystrokes.

The same operators (**FRom, TO**, <, >, and =) are used when searching in fields of type **NUmber**, **INteger** and **TIme**. A **NUmber** field contains one or more numerical values, an **INteger** field contains whole numbers (positive and negative), and a **TIme** field contains times of day expressed in hours, minutes, and seconds. The times in the **TIme** fields, like the dates in the **DAte** fields, need not be fully specified. The full form of a time is **14:20:10** (written in twenty-four hour time) for ten seconds past 2:20PM.

Searching in PHrase Fields

As mentioned earlier, searches are conducted only in fields containing data of type **TExt** or **PHrase** (the default), unless there are instructions to do otherwise. Fields of type **TExt** contain text consisting of paragraphs and sentences, while fields of type **PHrase** contain text of other kinds, such as short phrases, names, identifiers, and keywords. In the **Corr** database there are **PHrase** fields which contain information about the senders and recipients of the correspondence, which are repeated here for easy reference.

Field Name	Data Type	Contains		
rname	PHrase	Recipient:	name	
rcomp	"		company	
raddr	"	"	address	
rcountry	"	"	country	
sname	"	"	Sender:	name
scomp	"	"	"	company
saddr	"	"	"	address
scountry	"	"	"	country
day	DAte	Date of the message		
cat	PHrase	Type of documentation		

content	TExt	Text of the message
modified	PHrase	Modified by (TRIP username)
moddate	DAte	Modification: date
modtime	TIme	" time
modnote	PHrase	Modifier's note

Figure 2–10 The structure of database Corr

How do you find what you are looking for? If you are searching for correspondence to or from a person or a company named 'J Smith', but are not interested in correspondence that just mentions this name in the text of the correspondence, give the **Find** order:

```
f ph=j# smith ↵
Search  Records   Command
…
9       2         Find PHrase=j# smith
```

This order finds all terms in any **PHrase** field of the database where a word starting with 'J' is immediately followed by 'Smith', i.e. TRIP will look everywhere in the database except in the **day** and **moddate** (data type **DAte**), **content** (data type **TExt**) and **modtime** (data type **TIme**) fields.

If you are interested only in correspondence where a 'J Smith' is the sender, you need only search the contents of the field **sname**, or sender's name. Give the order:

```
f sname=j# smith ↵
Search  Records   Command
…
10      1         Find sname=j# smith
```

You can also use the symbols **<**, **>**, **<=** and **>=** to search for *intervals* in **TExt** and **PHrase** fields. For example,

```
f rname > eric ↵
Search  Records   Command
…
11      70        Find rname > eric
```

finds all records in **Corr** where the contents of field **rname** fall alphabetically after (that is, having an alphanumeric value in the collating sequence greater than) 'Eric'.

To specify both upper and lower limits in a search, insert two full stops (**..** symbol) between the search terms. Try this query:

```
f rname=p .. pz
Search  Records   Command
…
12      13        Find rname=p .. pz
```

Note:

A single space both before and after the interval symbol '..' is necessary.

This is the equivalent of the statement 'Find all recipients' first names beginning with *p*'.

In a large database, even a search order such as the one above may result in too many hits to readily evaluate. In these cases it is a good idea to take a look at the database vocabulary first, in order to see how many records contain the word stem, the word, or the phrase that you have in mind before you formulate the search order. The next section shows how to do this, and illustrates the difference between **TExt** and **PHrase** fields.

Displaying the Database Vocabulary

TRIP stores each single character (unigram), pair of characters (bigram), and character triplet (trigram), as well as each word or phrase as a unit in an *index* known as the data dictionary or vocabulary. The order **Display** (shortest form **D**) followed by a single term is used to view the database vocabulary for that word. Terms can then be selected from the vocabulary list that has been displayed and included in a **Find** order.

You can produce a list of the vocabulary of selected fields by including a field name or **PHrase** or **TExt** in the **Display** order. If no field names are specified, the list of vocabulary terms is taken from all **TExt** and **PHrase** fields by default.

Note:

> *These defaults are set at the start of a session, but new defaults may be defined at any point during the session. Refer to the 'Combination of Fields: Views' section in Chapter Five of this manual for more information.*

To see every term beginning with 'dev' in all **TExt** and **PHrase** fields, use the **Display** order:

```
d dev# ↵
```

The resulting **Display** window is shown:

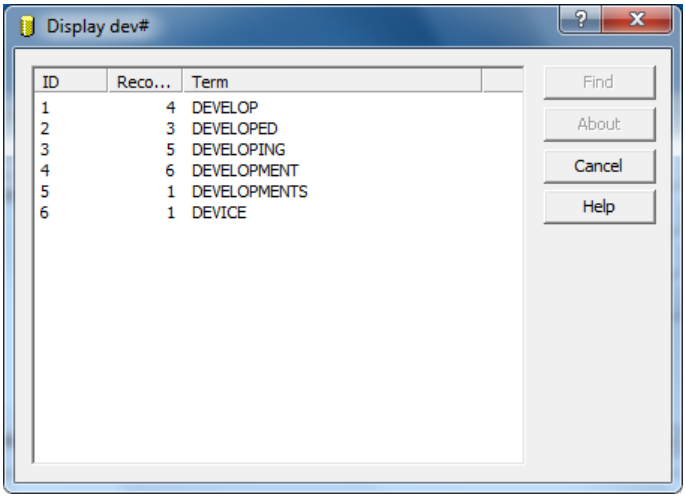

Figure 2–11 A Display list in the Display window

The Display list has a title bar labelled '(d dev#, 6 terms)', which shows the total number of terms which match the word stem given and the order used to generate it. As with **Show**, TRIP does not record your **Display** request in the History box as it did for **BASe** and **Find**.

Each of the terms found is labelled at the far left with an ID (1, 2, etc.). The number of records which contain each term is shown to the right of each ID.

To generate a search order using some of these terms, they must first be selected from the list. You may do this in one of two ways:

1. by including their term numbers directly in a Find order like this (we will use term numbers one through four in these examples):

```
f t=3 ↵

Search  Records   Command

…
13      5         Find "DEVELOPING"
```

TRIP writes the terms you have chosen in the History window.

You may also request TRIP to select more than one item in this way. Like this:

```
f t=1 to 4 ↵
Search  Records   Command

…
14      14        Find ("DEVELOP" OR "DEVELOPED" OR
                     "DEVELOPING" OR "DEVELOPMENT")
```

or like this:

```
f t=1,2,3,4 ↵
Search  Records   Command

…
15      14        Find ("DEVELOP" OR "DEVELOPED" OR
                      "DEVELOPING" OR "DEVELOPMENT")
```

TRIP will compile a list for you from either of those commands.

2. by (<Crtl> or <Shift>) clicking on individual terms within the display window and then clicking on the 'OK' button.

Note:

Holding down the <Ctrl> key whilst clicking will permit the selection of multiple terms and the <Shift> key will permit the selection of blocks of terms.

Select the terms with ID numbers one, two, three and four (1, 2, 3, 4) and click on the 'OK' button.

The **SElect** order builds the same search order as before:

```
Search  Records   Command

…
16      14        Find ("DEVELOP" OR "DEVELOPED" OR
                     "DEVELOPING" OR "DEVELOPMENT")
```

You can see that picking terms from a Display list is the same as giving a search order, with the terms combined by OR. Because some records contain more than one of the terms in the list, the number of records found will be less than the sum of the numbers of records for each term selected. If the list your request generates is longer than one screen can display, browse through the list using the vertical scroll bar that appears.

As shown by the preceding examples, a simple **Display** order presents a list of single words in alphabetical order. The remainder of this section covers the **Display** order in **PHrase** fields.

To search all the **PHrase** fields for the term ‘smith’, try the **Display PHrase** order:

```
d ph=smith ↵
```

The Display list, headed by the title bar ‘(Display PHrase=Smith, 4 terms)’ looks as shown in Figure 2 – 12:

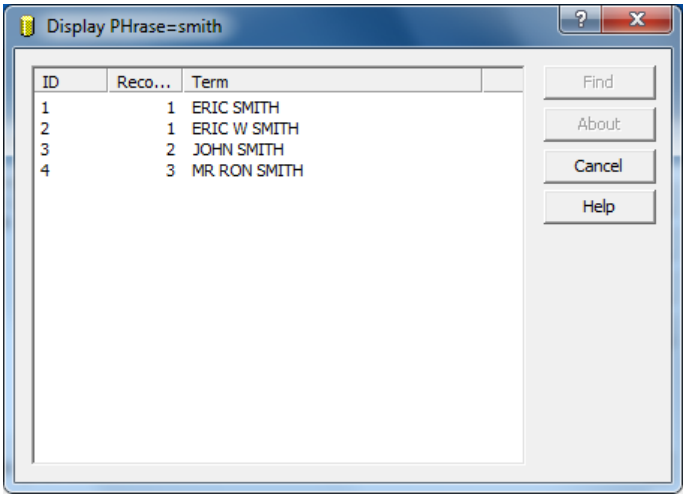

Figure 2–12 A Display list from a PHrase field

The Display list contains both names, although the order contained only the last name. This points to an important difference between **TExt** and **PHrase** fields.

While **TExt** fields consist of text divided into paragraphs, sentences, and words, **PHrase** fields are divided into subfields. Each subfield contains a short text that is treated as a unit, which in this case is the name of a person. When you request a display of the name ‘Smith’ in a **PHrase** field, TRIP shows the whole contents of all the subfields that contain the word ‘Smith’, first name, last name, middle initial, title or whatever has been entered.

Because short texts act as units, the **Display** order above could be more finely tuned by adding part of the first name to it:

```
d ph=j# smith ↵
```

TRIP shows only two records in the database in which ‘Smith’, first initial ‘J’, appears in a **PHrase** field. In this example ‘John Smith’ was the only phrase that matched the truncation pattern specified.

Note:

> *A **Display** order containing more than one term can be used only with **PHrase** fields. Because the vocabularies of **TExt** fields consist of single words, **Display** orders for **TExt** or for a combination of both **TExt** and **PHrase** fields can include only a single term.*

You can give a **Display** order for a single field as well, using the name of the field in your request instead of the field type:

```
d sname=smith ↵
```

This order gives a display of three Smiths (‘Eric W’, ‘John’ and ‘Mr Ron’) who sent letters included in this database. To find all contents of the field **sname**, use:

```
d sname=# ↵
```

TRIP displays thirty-eight phrases in the dictionary for this field.

Reports (Output Formats)

One or more reports can be predefined by the database manager when a database is created. How these look will depend on the database and the needs of the users.

The database **Corr** has two predefined reports:

- Format 1 (the default format) displays all of the information contained in the record except the contents of the field **sname** (sender name), as that information is also part of the text of the correspondence.
- Format 2 gives only sender and recipient information.

Both formats show the record number.

Type the order:

```
f sname=j# smith ↵
Search  Records   Command
...
17      1         Find sname=j# smith
```

which happens to be search number seventeen (S=17) in this session.

Type the **Show** order:

```
s s=17 ↵
```

or as search number seventeen was the last search performed, simply:

```
s ↵
```

This will show your search results in the fuller format (Format 1, since this has been defined as the default and we did not specify otherwise), as shown in the following figures.

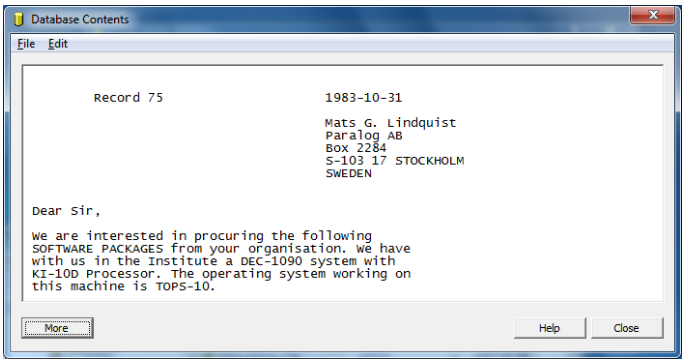

Figure 2–13 Format 1, screen one of two

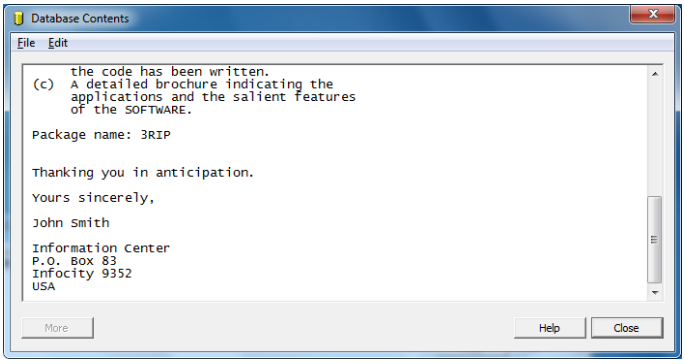

Figure 2–14 Format 1, screen two of two

You can select the format in which your search results will appear using **Format** (short form: **F**) in a **Show** request. In the order:

```
s s=17 f=2 ↵
```

'f=' (for 'Format=') is followed by the name of the predefined format (Format 2), which has been created by the database manager to display only the name, company, address, country, date of message and record number fields. In this format the body or textual content of each letter will not be displayed.

If search number seventeen was the last search performed, just:

```
s f=2 ↵
```

will show the record in the shorter format.

The output of this **Show** order is given below:

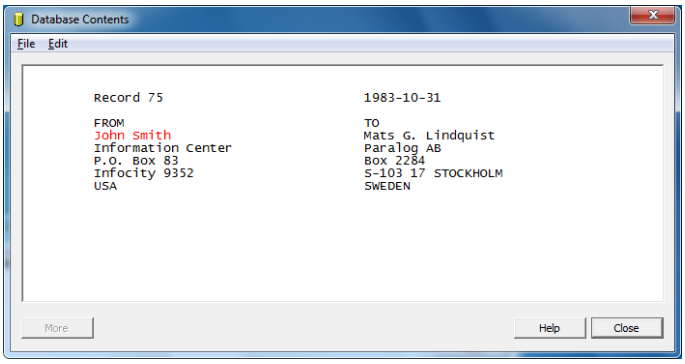

Figure 2–15 Format 2

A format requested in a **Show** order is volatile, and will be applied only to the search results displayed in that **Show** order. Every display thereafter will revert to whatever default format has been specified if there are no further instructions.

Since we are searching **PHrase** (and not **TExt)** fields, the name 'John Smith' is highlighted. To make this shorter format the default for future **Show** (and **Print**) orders, type the **DEfine** (short form: **DE**) order:

```
de f=2 ↵
```

TRIP acknowledges your request with the pop-up message, "Format 2 is now the default format." This means that where future **Show** (and **Print**) orders do not specify the format, search results will now be shown in Format 2. This default remains in place until another **DEfine Format** order is given, a new database is opened, or the TRIP session is ended.

If some other format is required, a selected set of fields forming a *run-time format* (one which is defined when a **Show** or **Print** is requested) can be specified. This can be done either in a **DEfine** order to make it the default, or directly in a **Show** order. Once again, run-time formats requested in **Show** statements will remain in effect only for that order. Subsequent **Show** (and **Print**) requests will revert to the default unless a new format is requested.

Suppose the only pieces of information you wish to see are the company name of the sender (field name **scomp**), the date (field **day**) and the content of the letter (**content**). To make this display the default for all subsequent **Show** and **Print** requests, give the **DEfine** order:

```
de f=scomp,day,content ↵
```

The message “New run-time default format defined.” appears in a pop-up.

To show only the contents of the **scomp**, **day** and **content** fields for just *one* search (rather than all of the searches performed during this search session), specify the format in the **Show** order as follows:

```
s s=17 f=scomp,day,content ↵
```

or if you wish to format the results of the last search performed, just use:

```
s f=scomp,day,content ↵
```

The results are shown in list form in the following figures, with the contents of each field headed by the name of the field.

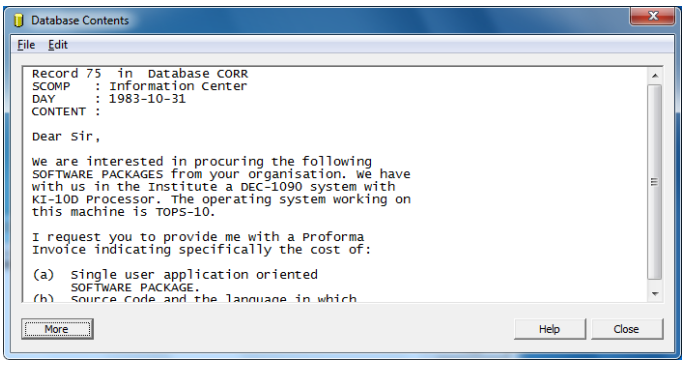

Figure 2–16 A run-time format, screen one of two

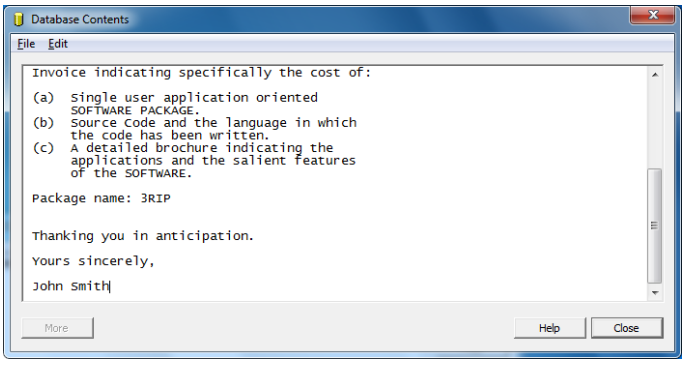

Figure 2–17 A run-time format, screen two of two

If it is difficult to remember the field name, you can substitute the field type for the name in a command.

For example, **content** is the only **TExt** (short form: **TE**) field in the **Corr** database. Try retyping the previous order, substituting data type **TE** for field name **content**:

```
s s=17 f=scomp,day,te ↵
```

This gives the same output as our previous **Show** request.

The field types can always be used in a format specification, alone or mixed with the names of individual fields. For review, they are (short forms are in parentheses): **TExt** (**TE**), **PHrase** (**PH**), **DAte** (**DA**), **TIme** (**TI**) and **NUmber** (**NU**).

Here is an order that contains all the field types:

```
s f=te,ph,da,ti,nu ↵
```

In this statement, **NUmber** includes fields of type **NUmber** and **INteger**. This shows all the fields of the records, headed by their field names, in the order in which they were specified. A **Show** order will be processed even if it contains a field type that is not represented in the database.

Previous Orders and Commands

When you give orders or commands in the CCL Search window, they are saved in a *history list* that is displayed in the 'Previous Commands:' drop-down selection box. You can reuse commands by selecting them from the drop-down list and immediately clicking on the 'Query' button, or you may edit and reuse them.

Before we continue with the tutorial, make a new search for senders of correspondence with the first or last name beginning with 'Sven', like this:

```
f sname=sven# ↵
Search  Records   Command
…
18      4         Find sname=sven#
```

Then define a new **Show** default to include only fields **sname** and **day**:

```
de f=sname,day ↵
```

Sorting the Results

Quite often it is useful to be able to **SORt** (short form: **SOR**) the result records in a particular order before showing or printing them. You could, for example, sort a list of results alphabetically by name of sender, using the order:

```
s sor=sname ↵
```

Try this now. You should have one record containing 'Jan *Sven*sen', two with 'Orvar *Sven*sson' and one with 'Ulf *Sven*dsen', all from the most recent search performed (S=18).

To **SORt** on a date field, try this order with **day**:

```
s sor=day ↵
```

Note:

> *Since a SORt can only be requested in a Show or Print order, TRIP does not note your action in the History area.*

The records are now sorted in normal chronological order from 1984-01-10 to 1984-10-30 (*ascending* or increasing order is the default).

It is also possible to sort on a combination of fields. Use S=18 once again and the following search order to sort all the records first by sender's name (in this case by the first name, since last, first, middle initial, title etc. are treated as a unit here), then by date:

```
s sor=sname,day ↵
```

To see a search result sorted in *descending* or reverse order, add the modifier **DEScending** (short form: **DES**) after the field name. In **Show** orders containing multiple **SORt** requests (as does the order above), the sorts will be *nested*.

For example, in the search above TRIP will sort first by **sname**, then by **day** within **sname**, and since there are multiple hit records only for 'Orvar Svensson', these are the only records that will *appear to be* sorted by date.

Try the following orders:

```
s sor=sname,day des ↵
s sor=sname des,day ↵
s sor=sname des,day des ↵
```

In the first example, **sname** is sorted in ascending order ('Jan', 'Orvar', 'Ulf') and **day** in descending (remember, within 'Orvar' only, 1984-06-15 to 1984-01-10).

The second statement sorts **sname** in descending order ('Ulf', 'Orvar', 'Jan') and **day** ascending (in 'Orvar', 1984-01-10 to 1984-06-15). The last example sorts both **sname** and **day** in descending order.

As before, requests for descending sort order in **Show** and **Print** orders are volatile and must be respecified with each statement.

Databases and Defaults: STatus, BASe and DEfine?

As we saw in the Introduction, if the order **BASe** is given without a database name a list of all the databases that may be opened by you are shown, together with the description of their contents. This is useful when you cannot remember the name of a database, or cannot remember which of a number of databases contain the records you wish to search.

The command **STatus** (short form: **ST**) followed by the name of a database gives general information about that database. If a database is already open, the command **STatus** alone gives the same information about the open database. If you have more than one database open at any one time, a **ST** ↵ order will present status information for each open database, in alphabetical order by database name.

The order:

```
st corr ↵
```

or if **Corr** is already open (as now), simply

```
st ↵
```

presents screens such as the following:

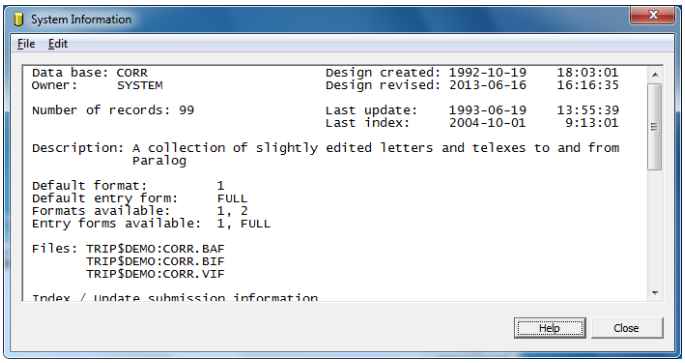

Figure 2–18 STatus Corr, screen one of three

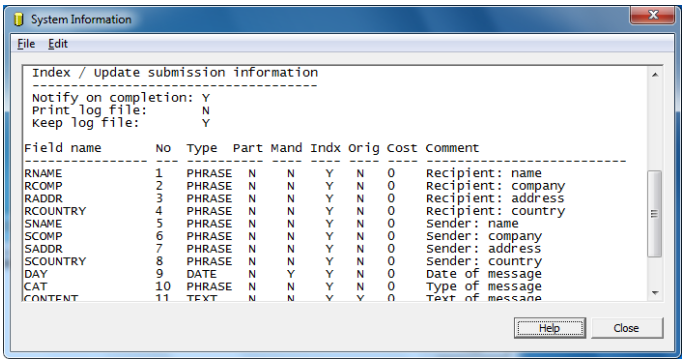

Figure 2–19 STatus Corr, screen two of three

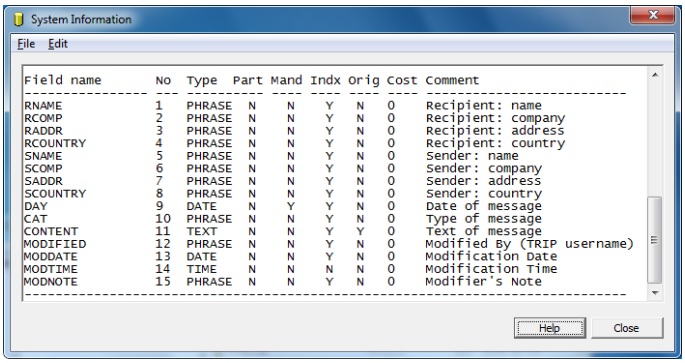

Figure 2–20 STatus Corr, screen three of three

Reading from left to right on the first **STatus** screen, the following information is available (dates and times will differ with each installation):

- the database name (**Corr**)
- the creation date (October 19, 1992) and time (10:03AM)
- the owner/creator (System)
- the date and time of design revision (May 23, 1993, 10:41PM)
- the number of indexed records in the database (99)
- the dates and times when new material was last added to the database (‘Last update: October 19, 1992, 2:20PM’) and made searchable (‘Last index: May 24, 1993, 8:47AM’)
- a brief description of the database contents
- a ‘forms and formats’ list giving the names of the default format, default entry form and other reports and entry forms that are available
- a list of TRIP filenames used by the database manager for this database (this may continue onto the next screen).
- The second and third **STatus** screens present:
- some miscellaneous information used by the application manager
- a field list containing:
- the name, field number and data type of each field
- columns labelled ‘Part’, ‘Mand’, ‘Indx’, ‘Orig’, ‘Cost’ and ‘Comment’:

- 'Y' or Yes in the 'Part' column indicates that field possesses a special bi-level field structure called a *part field* (see Chapter Eight, 'Head And Part Records' in this guide for more information)
- 'Y' or Yes in 'Mand' indicates that this field is a '*Mand*'atory field, that is, never empty in a record
- 'Y' or Yes in the 'Indx' column indicates that the field is indexed. It is not possible to search in or display the contents of a field which contains 'N' in 'Indx' (that is, not indexed), although the field contents can be shown if the record is found by searching in other fields
- 'Orig' (short for '*Orig*'inal Layout) specifies whether or not TRIP will store the information in its original form (complete with tabs, spaces and other formatting instructions) rather than in compressed form
- 'Y' or Yes in the 'Cost' area indicates an expense or monetary outlay when one **Show**s or **Print**s the contents of this field (there is, however, no charge for searching these fields). You can view a summary of any charges you have incurred during any particular search session using the command **Show COST** ↵
- 'Comment' contains a brief description of the field itself.

We have already seen that the order **DEfine Format** can be used to define the default report. The **DEfine** command can also be used to define other defaults. To view a list of system defaults, as well as new defaults that may have already been specified, use the **DEfine?** (short form: **DE?**) order:

```
de? ↵
```

A possible list of defaults and their explanations is shown in the accompanying table. For more information regarding the various **DEfine** commands, refer to the *CCL Command Reference*.

DEfine? Term	Meaning
Highlight = All	All hit terms in output are highlighted
No focus	Full hit information is shown in output
No merge	Records will be sorted within each database
No reverse	Records shown in ascending order
Hold	Print orders will be assigned to a queue
Save base	BASe orders will be saved when Save order is given
Tstamp Update	A timestamp search of records, excluding updated records, is enabled
No stop word	No stop words are defined for the CCL **FUZz** command
Display no orig	DISPLAY lists will use original forms of the terms
Display freq = merge	Merging across databases of terms in frequency restricted term lists is enabled

Find = no Fuzz	Find command will not behave as if it were the Fuzz command
Page	In TTY (TeleTYpe) mode, screen output is non-continuous, displaying one page of data at a time. Use More to scroll to next screen
FIND max = no limit	No limit on the number of search terms allowed in any individual query
+ max = no limit	No limit on the number of search terms that can be joined by + in any one query
DISPLAY max = 1000	Can display up to and including 1000 terms per display order
SORT max = 1000	Can sort up to and including 1000 records per search request
MAP max = 1000	Can search up to and including 1000 terms using a 'virtual field'
DELETE max = 1000	The global record delete has a maximum value of 1000 defined
PRINT max = No limit	Can print any number records at any one time
AND = AND.E	The logical AND denotes 'AND within the same record entity' while searching in records that contain head/part records
DEfine? Term	**Meaning**
MASK = '#:!&'	Four masking symbols are currently available for use
TIMEFORM = 1	Format for TIme fields is hh:mm:ss
CENTURY MIN = 1953	If the two digits a user has supplied for the year are 53 or greater, then 1900 will be added to the year, otherwise 2000 will be added
FUZZ = 75, 5, 2, 1	Fuzzy searching will be according to the listed parameters.
ABOUT = 50, No Highlight	Precision in matching is 50% and hit highlighting is off for non-Boolean searching
VIEW=TExt, PHrase	Only the information stored in Text and PHrase fields will be searched
KEY	No function keys are defined as soft function keys or macro keys

Table 2–21 List of DEfine defaults

Error Messages and Help

If you give an order that TRIP cannot accept or is unable to interpret, an error message will appear in a pop-up message box, and the erroneous order will be left in the Command box for correction or deletion.

Other kinds of messages from TRIP, such as those confirming that a new default format is set or that some other action has taken place that does not lead to an immediately visible result, appear in pop-ups as well.

TRIPmanager provides help on the use of all of its commands and their modifiers and operators, and is accessed from the ‘Help’ menu on the mmc menu bar. Contextual help, relating to a selected item, is also available from the action menu.

Ending a Search Session

To exit CCL, close the CCL command window by clicking in its ‘Close’ button.

Note:

For the next tutorial, if you close the current session,, you will need to re-enter the searches currently in Search History. They are listed at the beginning of the tutorial for your convenience.

Chapter 3:
Tutorial Three

This third search session shows how to search for records which exclude certain terms, how to search using record numbers and how searches may be combined with previous searches.

It shows how to focus in on hit terms, and skip from one hit to the next. It also shows how to change databases during a search session, how to save a search order and rerun it, and how to delete searches.

It ends by giving more examples of the truncation of words and of *masking characters*, symbols that stand for some unknown letter or letters.

We saved all searches and ended the last session by giving the order **STOP SAve**, so we will begin this tutorial with the search orders which have been stored in CCL History from that session.

You will need to select the **CCL Query** option for database **Corr** now. You should enter the searches 2 to 18, so that they appear in your history window as below:

Search	Records	Command
1	99	BASe corr
2	19	Find price OR lease
3	17	Find price XOR lease
4	14	Find price NOT lease
5	12	Find S=2 AND day > 1984-06-30
6	12	Find S=2 AND day >= 1984-07
7	12	Find S=2 AND day=FRom 1984-07
8	9	Find S=2 AND day=1984-07 FRom 1984-10
9	2	Find PHrase=j# smith
10	1	Find SNAME=j# smith
11	70	Find rname > eric
12	13	Find rname=p .. pz
13	5	Find "DEVELOPING"
14	14	Find ("DEVELOP" OR "DEVELOPED" OR "DEVELOPING" OR "DEVELOPMENT")
15	14	Find ("DEVELOP" OR "DEVELOPED" OR "DEVELOPING" OR "DEVELOPMENT")
16	14	Find ("DEVELOP" OR "DEVELOPED" OR "DEVELOPING" OR "DEVELOPMENT")
17	1	Find sname=j# smith
18	4	Find sname=sven#

If they are not already there, type them in now.

Notes:

- *If you are viewing this document electronically, we suggest you copy each of the commands (2 to 18) one at a time, from the right hand column above, into the CCL command entry box (without the two left-hand number columns), where each command may then be executed:*

- *In this document, the commands 14, 15 and 16 wrap across two lines but, as there is no linefeed/carriage return character, they may be copy pasted with the included tabs as TRIP will treat this whitespace as a single space.*

Combining With the Latest Search Result

It is a general rule in TRIP that the latest search result performed is the default (that is, the search result acted upon) when no specific search result is given. This was demonstrated in the last session using the **Show** order. If you type the order:

f and new ↵

TRIP will use the result of the last search stored in History (S=18) to find all records where the first or last name of any sender begins with 'sven' and which contain the term 'new'. This example appears in the search history as:

```
Search  Records   Command
…
19      3         Find S=18 AND new
```

This is because the modifier **AND** must have a search term on either side of it, so TRIP inserts a reference to the latest search result. Another way of referring to the latest search is S=0, so the search just completed could be rewritten in this way:

f s=0 and new ↵

```
Search  Records   Command
…
20      3         Find S=19 AND new
```

This example, as well as every other example in this chapter, shows the Search History notation immediately following each CCL order.

Deleting Searches

Any search result can be removed, if, for example, it produced no useful results and is cluttering the History window. A search result is removed using the command **DELete** (short form: **DEL**), and as the latest result is always the default, it can be removed simply by giving the order:

del ↵

Type that order now.

Search result number twenty (S=20) will be deleted, and result number nineteen (S=19) becomes the last or most recent result.

Note:

Deleting searches from the middle of the Search History will not cause all of the remaining searches to be renumbered, but subsequent searches will be added on to the search history at the correct number position!

Repeat **del ↵** again to remove search result nineteen.

This type of **DELete** order affects only search results, not the records in the database itself. Special privileges are required to make changes in the database, and the syntax of that **DELete** order is quite different.

‘Negative’ Searches

This section covers the use of the **NOT** modifier, which finds all the records in the database that exclude (that is, do not contain) a specific term.

NOT works similarly to **AND**, but may lead to some search results one might not ordinarily expect.

Enter the order:

```
f not new ↵
Search  Records   Command
…
19      1         Find S=18 NOT new
```

TRIP treats this order in the same way as the **AND** order discussed previously, finding all records containing ‘sven#’ in **sname** but *which do not contain* ‘new’.

You might ordinarily expect this order to find every record in the database which does not contain the word ‘new’. However, TRIP assumes that you wish to look for ‘new’ *only in the results of the last search performed*, therefore this order searches only the records found by S=18.

Using another example, to find all of the correspondence in **Corr** that was not sent by Paralog (about half of the records in the database), search the field containing the company name (**scomp**) like this:

```
f all not scomp=paralog ↵
Search  Records   Command
…
20      37        Find ALL NOT scomp=paralog
```

TRIP interprets this search order as ‘find ALL the records *except* (that is, **NOT**) those where the term ‘paralog’ appears in the field **scomp**’. Since ALL was included in the command, TRIP does not limit the search to the last search result obtained (S=19).

Searching for Empty/Non-Empty Fields

In certain circumstances, you may wish to find all of the records for which a certain field is empty, or conversely, the records for which a particular field contains some terms.

There is a field named **cat** (Category) in the **Corr** database that is of data type **PHrase**. Give the order:

```
f cat=# ↵
Search  Records   Command
…
21      3         Find cat=#
```

to find all records (three hits) containing some value in **cat** (that is, where **cat** is not empty).

To search for records where the field **cat** *is* empty, use the order:

```
f cat="" ↵
Search  Records   Command
```

```
…
22      96          Find cat=""
```

Note:

There is no space between the double quotes.

Ninety-six records have no data in **cat**.

The two orders are interpreted by TRIP as follows:

- find all records where **cat** contains **#** (meaning anything, but not nothing)
- find all records where **cat** contains **“ ”** (the empty string, meaning nothing).

Searching Using Record Numbers

Every record in a database has a *record number* (short form: **R**) or unique record identifier, which is never duplicated or reused (in the case of record deletion) in that database. These can be used for searching in this manner:

```
f r=fr 49 ↵
Search  Records   Command
…
23      51        Find r=FRom 49
```

TRIP interprets this as ‘find all records with a record number greater than or equal to 49’.

If we then follow this with:

```
f not scomp=paralog ↵
Search  Records   Command
…
24      17        Find S=23 NOT scomp=paralog
```

TRIP will assume that we wish to base this search upon the last search result (S=23) obtained.

This finds all the records between (and including) record numbers forty-nine and ninety-nine (S=23) which do not have the term ‘paralog’ in the field **scomp** (S=24).

We can also combine record numbers with other search conditions in **Find** statements such as this:

```
f r=fr 49 not scomp=paralog ↵
Search  Records   Command
…
25      17        Find (r=FRom 49) NOT scomp=paralog
```

which also finds all the records numbered forty-nine and above that do not have the term ‘paralog’ in the field scomp.

FOcus - Showing Only Hits in Records

The FOcus modifier (short form: FO) is used after the Show command to restrict the output of the records hit. In TExt fields, only the paragraph in which the hit or hits occur is displayed. This is especially useful in long records, where you can move directly to the paragraphs containing the subject matter of interest.

To illustrate, we will use the last technique described to find all of the occurrences of the word 'is' in record numbers forty-nine and fifty:

```
f r=49,50 and is ↵
Search  Records   Command
…
26      2         Find (r=49, 50) AND is
```

Note:

*The comma here is interpreted as **OR**.*

To see *only* the first paragraph containing a hit (with no surrounding text), use **Show FOcus** (short form: **S FO**):

```
s fo ↵
```

TRIP gives you some information at the top of the **Show** screen that is designed to help orient you within your search result:

- the record number within the hit record count (Record 1 of 2)
- the name of the database you are currently searching (**Corr**)
- the record number of the record you are examining in the database (.R=49)
- and the field name containing your search target (**content**).

Below that will be printed the portion of the hit record containing the search term you are interested in, without any extraneous information to distract you. You may scroll the records as with any other output.

The difference between **Show FOcus** and **Show Format=*fieldname*** (as discussed in Tutorial Two) is that when you **FOcus** a hit record, you are merely blanking out or hiding extraneous information, both within and outside the field in question. This leaves large gaps or blank spaces in the screen display in the process. When you **Format** a hit, however, you are actually displaying only the fields you are interested in, and all available information within these fields will be shown.

Once you have focused on the hit paragraph, you can return the hit record to its original form and view the paragraph in context by using the **Expand** feature (short form: **E**) to restore the information around the hit. To **Expand** the text, type

```
e ↵
```

All text which normally surrounds the hit paragraph appears in a new window, displaying the entire record. You can scroll up and down through the new screen output in the usual way.

All the hits in this example were found in paragraphs of the field **content**, which is of type **TExt**. If the hits are made in fields of any other type (**PHrase**, **DAte**, **TIme**, **NUmber** or **INteger**), it is the content of each subfield, rather than each paragraph, which is shown.

For example, try typing the search order:

```
f saddr=5# ↵
Search  Records   Command
…
27      3         Find saddr=5#
```

where we are looking for any term in the Sender Address field beginning with '5'.

Now **FOcus** the hits:

```
s fo ↵
```

TRIP prints only the line of the address (the subfield) in which a leading '5' was found (P.O. Box ***53,*** Edf Caramba Apt ***59C***, ***53*** Eden Street).

Saving a Search Order

When ending a search session or changing to another database, you may want to keep one or more of the search orders you have made in order to reproduce the result in a later session. This is done by saving a search order and giving it a name (maximum length, sixteen alphanumeric characters, including underscore).

At this point your search history should contain these search results:

```
S=1     <99>      BASe Corr
S=2     <19>      Find price OR lease
S=3     <17>      Find price XOR lease
S=4     <14>      Find price NOT lease
S=5     <12>      Find S=2 AND day > 1984-06-30
S=6     <12>      Find S=2 AND day >= 1984-07
S=7     <12>      Find S=2 AND day=FRom 1984-07
S=8     <9>       Find S=2 AND day=1984-07 FRom
1984-10
S=9     <2>       Find PHrase=j# smith
S=10    <1>       Find sname=j# smith
S=11    <70>      Find rname > eric
S=12    <13>      Find rname=p .. pz
S=13    <5>       Find "DEVELOPING"
S=14    <14>      Find ("DEVELOP" OR "DEVELOPED" OR
                    "DEVELOPING" OR "DEVELOPMENT")
S=15    <14>      Find ("DEVELOP" OR "DEVELOPED" OR
                    "DEVELOPING" OR "DEVELOPMENT")
S=16    <14>      Find ("DEVELOP" OR "DEVELOPED" OR
                    "DEVELOPING" OR "DEVELOPMENT")
S=17    <1>       Find sname=j# smith
S=18    <4>       Find sname=sven#
S=19    <1>       Find S=18 NOT new
S=20    <37>      Find ALL NOT scomp=paralog
S=21    <3>       Find cat=#
S=22    <96>      Find cat=""
S=23    <51>      Find r=FRom 49
S=24    <17>      Find S=23 NOT scomp=paralog
S=25    <17>      Find (r=FRom 49) NOT scomp=paralog
S=26    <2>       Find (r=49, 50) AND is
S=27    <3>       Find saddr=5#
```

In order to **SAve** (short form: **SA**) your last search result and the statements that produced it, use the command:

```
sa last_search ↵
```

where ‘Last_Search’ is a TRIP *procedure* or miniature program containing the search result. You will receive the message, ‘Search 27 is saved as LAST_SEARCH’.

To **SAve** the CCL commands that produced search result number nineteen under the name ‘Sven_Not_New’, give the order:

```
sa s=19 sven_not_new ↵
```

TRIP responds with the message, “Search 19 is saved as SVEN_NOT_NEW”.

TRIP saves just enough of the search history to reproduce the result; that is, it saves all of the searches on which the result depends, but none of the others. Since only

three statements were necessary to produce the single record found by search number nineteen, 'Sven_Not_New' will contain only these statements:

```
S=1     <99>      BASe corr
S=18    <4>       Find sname=sven#
S=19    <1>       Find S=18 NOT new
```

To produce this search result again during another session, simply type the name of the search sequence you have saved. Run 'Sven_Not_New' on the CCL Command line now using:

sven_not_new ↵

As TRIP executes the saved commands, each will appear very briefly in sequential order in the Command window. Each result will be recorded in the History box in the usual way.

TRIP can also save an entire search history. To save *all* of the searches currently in your History window under the name 'Third_Session', type:

sa s=1 to 27 third_session ↵

TRIP confirms your request with the message "Specified searches are saved as THIRD_SESSION".

Note:

*TRIP actually saves the search order as a private **PRocedure**. If required, you can edit and modify it to produce a different search result. See Chapter Eleven, 'User Procedures' of this manual for more information.*

If you would like to save a series of CCL commands without their corresponding **BASe** order(s), use the **DEfine SAve** (short form: **DE SA**) order:

de sa no bas ↵

The message is "Database search not included in saved searches." **BASe** commands will not be included in your saved search procedures until either you end your TRIP session or you use the command

de sa bas ↵

after which you will receive the message "Database search included in saved searches."

If you no longer need a saved search order, you can delete it using **DELete SAve** (short form: **DEL SA**). **DElete** 'Sven_Not_New' in this manner:

del sa sven_not_new ↵

TRIP gives this message: "Procedure SVEN_NOT_NEW deleted."

You can also use the command **DELete PRocedure** (short form: **DEL PR**). Discard 'Third_Session' now with this method:

del pr=third_session ↵

TRIP responds with "Procedure THIRD_SESSION deleted."

Deleting Searches, Changing Databases

You can delete all of the searches in your current History box with the **DELete Search** (short form: **DEL S**) order:

```
del s=all ↵
```

To clear History for the next exercises, give this order now. TRIP responds with "All sets deleted."

Note:

*Although the history window is not updated, all subsequent searches will be numbered from one onward, unless you perform another **DELete**.*

You may wish to discard all previous searches routinely when opening a new database, as earlier searches for a different database are unlikely to be useful. However, preceding searches may be helpful if the same database were reopened later in the same session.

You can use the **BASe** command again when you are ready to open another database. **BASe** simultaneously opens the new database(s) and closes all database(s) previously in use, or you can simply close the current CCL command window and open a new one for the new database.

The next examples are taken from another demonstration database called **Alice**. This database contains portions of Lewis Carroll's books 'Alice in Wonderland' and 'Through the Looking-Glass'—stories concerning a small girl who falls down a rabbit hole, steps through a drawing-room mirror and experiences unusual adventures with a variety of nonsensical characters.

Each record in the **Alice** database consists of a piece of the text, the name and number of the chapter from which the text is taken, and the names of the persons mentioned in the text.

Open a CCL Query session to this database now. Once the new CCL command window is open, you should see the following in the search history:

```
Search  Records   Command
1       475       BASe alice
```

STatus Revisited - On Screen and In Print

In Tutorial Two we discussed browsing the organization of a database before searching to become familiar with certain of its attributes, such as the size of the database (number of searchable records) and the field names, data types and formats that are available. Ask to see this information for database **Alice** now using **STatus**:

```
st ↵
```

The screens shown below will appear (remember that dates and times of database creation and revision are unique to every installation):

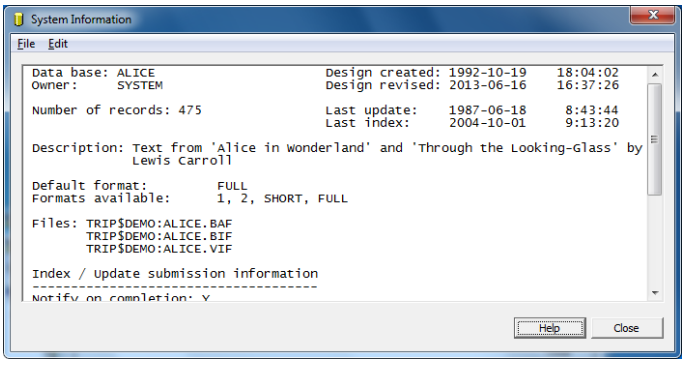

Figure 3–1 STatus order for Alice, screen one of two

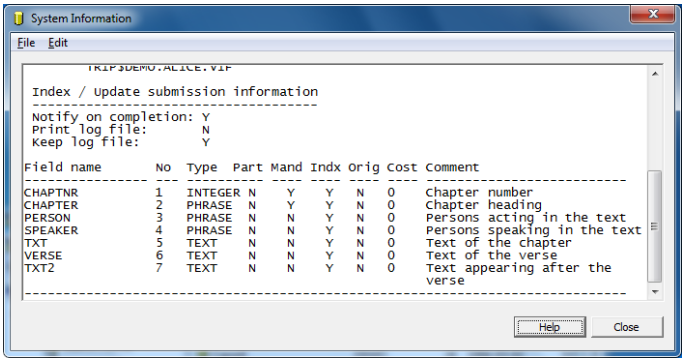

Figure 3–2 STatus order for Alice, screen two of two

Familiarize yourself with the field names and data types in this database. There are three **TExt** fields in **Alice**, **txt**, **txt2** and **verse**. The field **verse** contains the poems of the two books; however, the text is not indexed and thus cannot be searched. **Txt** contains the general text, while **txt2** includes text that follows a poem in a record. There are also two 'Mandatory' fields in every record in this database, the name (**chapter**) and number (**chaptnr**) of each chapter.

You can refer to the field name list at any point during the following exercises using

```
st alice ↵
```

Searching for a Whole Phrase

We have mentioned before that the purpose of a **PHrase** field is to store *phrases*, collections of words which naturally belong together and which usually have a meaning over and above the sum of their parts (for example, the phrases 'The Mad Hatter' and 'The Queen of Hearts').

A **PHrase** field can hold many individual phrases and usually stores them in subfields, each of which has a maximum length of 255 characters.

In order to match a search expression to the whole content (all words) of a subfield of a **PHrase** field, place single quotation marks around the expression in the search order.

Note:

Limiting an entire phrase is the only time single quotation marks are used in TRIP. In all other cases, double quotation marks are preferable.

Remember that the field **person** contains the names of persons acting in the text. Give this search order:

```
f person='queen' ↵
Search  Records   Command
1       475       BASe alice
2       3         Find person='queen'
```

and now this one:

```
f person=queen ↵
Search  Records   Command
…
```

```
3       115       Find person=queen
```

The first search order (three hits) finds only subfields of **person** which contain 'queen' alone. The second (115 hits) locates records where the subfields contain the term 'queen' alone or in combination with other terms ('Queen', 'Queen of Hearts', 'Red Queen', 'White Queen'). We can see this most easily by giving the **Display** order

d person=queen ↵

This list of four terms is shown in Figure 3 – 3:

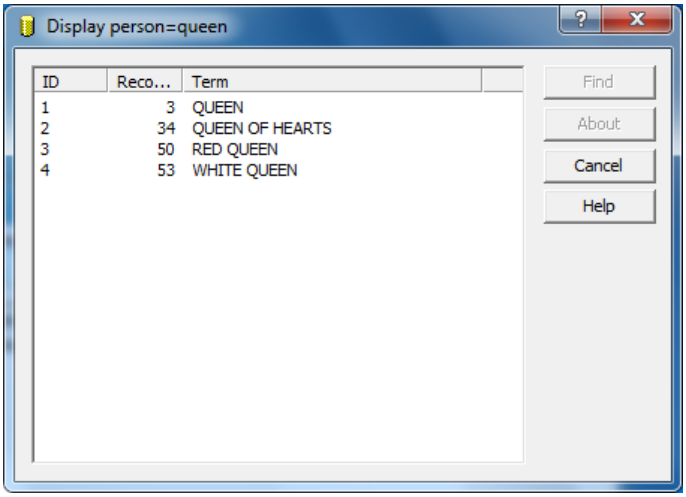

Figure 3–3 Display of Terms for PERSON=Queen

Subfields of a **PHrase** field are separated by commas when they are output using a run-time format. Give the **Show Format** order:

s f=person ↵

This format shows only the record number and the contents of **person** for each hit of the last search performed (S=3). Each value for **person** is separated from the next with a comma:

```
Record 83 in Database ALICE
PERSON : Fish Footman, Frog Footman, Duchess, Queen
```

Note:

*Any field name can be truncated (shortened) to a minimum number of characters which uniquely identifies it to TRIP. In fact, as there are no other field names beginning with 'p' the field name **person** could be shortened to its first letter only, making this the shortest form of the preceding order:*

s f=p ↵

Searching by Field Name

From the display list above, we can see that since the sum of the terms is greater than the number of records found (in this case, 115), some records must contain more than one of the terms shown. For example, to see how many records have both 'Red Queen' and 'White Queen' in their **person** fields, give the order:

f p=(white queen and red queen) ↵

```
Search  Records   Command
```

```
…
4       25          Find PERSON=(white queen AND red
queen)
```

The parentheses indicate that both terms apply to the **person** field. Without parentheses, the search order,

```
f p=white queen and red queen ↵
Search  Records   Command
…
S=5 <26>  Find PERSON=white queen AND red queen
```

would read ‘Find the records where the **person** field contains *White Queen*, and the term *Red Queen* appears anywhere in the record’.

More About Truncation and Masking

In the first tutorial, we introduced the truncation symbol ‘**#**’ to indicate an unknown number of characters at the beginning, the middle or the end of a word.

The colon truncation symbol [**:**] is a more precise way of omitting characters, and signifies ‘one letter or none’. This is useful when looking for a word in either the singular or plural. For example,

```
f cat: ↵
Search  Records   Command
…
6       25          Find cat:
```

finds twenty-two occurrences of ‘Cat’ and six of ‘Cats’ in twenty-five records, whereas the order:

```
f cat# ↵
Search  Records   Command
…
7       54          Find cat#
```

finds more than twice as many records.

If you **Display** the vocabulary generated by the latest term by giving the order

```
d cat# ↵
```

you will see that in addition to ‘Cat’ and ‘Cats’, this truncation locates thirteen occurrences of ‘Catch’ and ‘Catching’, nineteen of ‘Caterpillars’ and one incidence of ‘Cattle’.

Another useful symbol is the dollar sign [**$**], which replaces whole words. Look for the expressions ‘the little busy bee’ and ‘caterpillar’ by giving the order:

```
f the $ $ bee and caterpillar ↵
Search  Records   Command
…
8       1           Find the $ $ bee AND caterpillar
```

where each ‘**$**’ replaces one word (note the single space between the $$’s).

For more details about truncating and masking, refer to the **Find** command sections in the *CCL Command Reference*.

Part 2:

Advanced Searching

Chapter 4:
TRIPclassic Search Forms

The TRIPclassic search form is not currently available for use in searching with TRIPmanager. It is possible, however, to view the existing search forms in a TRIP installation and to carry out some administrative functions on those forms.

To see which TRIPclassic search forms currently exist on a TRIP installation, select the 'Search Forms' icon in the mmc window. A list of search forms will then appear:

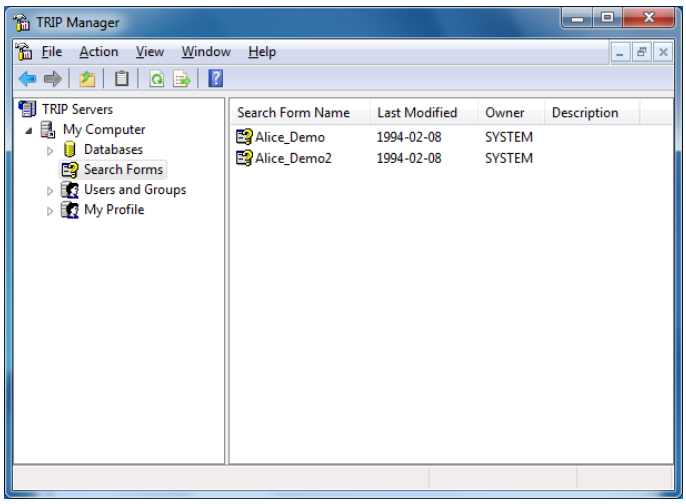

Figure 4–1 Search forms for a TRIP installation

Selecting a search form and choosing 'Properties' from the action menu, will list the properties for that form.

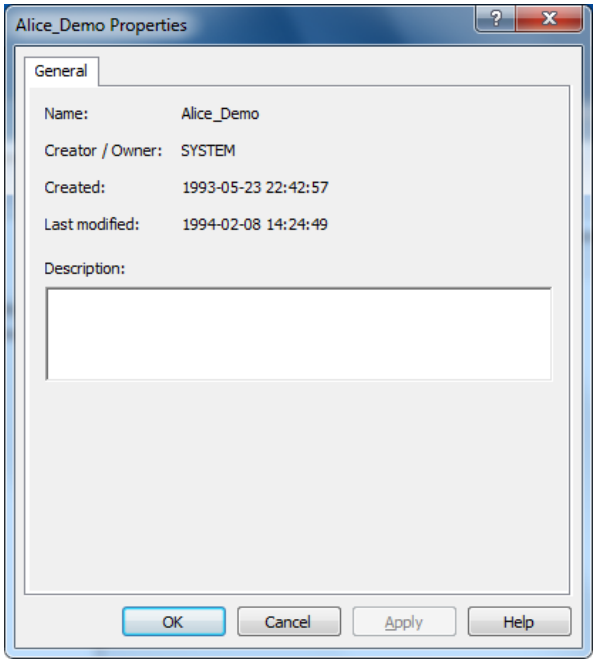

Figure 4–2 Properties for search form ALICE_DEMO

For further details on administering TRIPclassic search forms with TRIPmanager, consult the TRIPmanager Administration guide.

Chapter 5:
Advanced Features

This chapter contains advanced features of TRIP that are not covered by the tutorial sessions. Some sections include concepts which are introduced here for the first time, while others elaborate on features which have already been mentioned and are presented here in greater detail. Further references for each CCL statement are available in the *CCL Command Reference*.

Open a CCL query window for the demonstration database **Alice** to work the exercises in this section. You should see the following in the search history:

```
Search  Records   Command
1       475       BASe alice
```

All examples which include History results in this chapter use the format displayed in the example above.

Web Style Searching – Defining Find as FUZz

In TRIP, the default search style for the **Find** command is Boolean search which, for most users, is more than sufficient to meet their needs; however, there are times when different types of non-Boolean searching may be necessary and, traditionally, TRIP has had the **FUZz** command to provide such search capabilities.

Nonetheless, demand has risen for users to be able to use non-Boolean searching as standard and to this end, TRIP will permit the defining of the **Find** command so that it instead maps to the **FUZz** command in order to give a 'Web style' search facility that may be more familiar to some users than Boolean search.

For more information on how to achieve this, consult the *CCL Command Reference*, **Define Find (FUZz)** section.

Best Match Searching – Find ABout

Note:

Best match searching is only available for fields of type TExt or PHrase.

Overview

Whereas Boolean searching focuses on matching a request with a set of provably valid responses, best match searching focuses on matching a request to a set of responses that reflect the underlying intent of the request. From this set of possible matches, the search algorithm then provides a weighting (or judgment) as to how relevant each matched document is when compared to the information need expressed in the search query.

Performing a best match search

As shown below, there are many different ways of exercising the best match search function. The simplest is to state an information need explicitly, i.e. to give the function the text of the query that you wish to match. This query text could be a simple phrase, a full sentence, an arbitrary fragment, or an entire page of text:

```
Find ABOut(peanut butter)
```

Likewise, one or more terms from a Display list may be submitted as an information need, using the "T=" syntax:

```
Find ABOut(T=n [TO m])
```

Someone wishing to "find more like this one" would make use of the "R=" syntax, whereby the content of the specified record is analyzed for information need:

```
Find ABOut(R=n [TO m])
```

Finally, in certain constrained circumstances it may be useful to find documents that match the information need expressed by all records in one or more search sets, using the normal "S=" syntax:

```
Find ABOut(S=n [TO m])
```

For those of you wishing to find out more about best match searching, please consult the '**DEfine About**' and '**Find About**' sections in the *CCL Reference Manual* and read the TRIP white paper entitled, "TRIP_White_Paper_Non_Boolean_Search.pdf", which can be found in the 'doc' directory of the TRIPsystem installation.

Note:

Using best match searching on large search sets will result in extremely poor performance.

Setting up a database to support best match query

In either of the latter two cases in the previous section, the search algorithm will need to know what attributes of the records should be analyzed when attempting to construct the information need which is being expressed by those records. Equally, when indexing documents that will be tested for best match during queries of any of the forms shown above, the indexing engine needs to have this same information available.

The manager of such a database, therefore, must establish a new indexing flag on any field that is to be included in a non-Boolean calculation (either at indexing or query time). This is performed via TRIPmgr in one of two different ways:

Single field assignment is achieved by using the context menu on that field and setting the appropriate flag in the field's property page, as shown in Figure 2, overleaf.

Multiple field assignment is achieved by selecting all fields to be modified, and using the appropriate option from the context menu, as shown in Figure 3, overleaf.

For more information on how to achieve non-Boolean inclusion for fields, please consult the '*Create word-based index*' sub-section of '*The Modify Fields Collection Form*', in Chapter 2 of the *TRIPmanager Administration Guide.*

Searching using categories – Find Class()

It is possible to implement classification searching in TRIP; i.e. The searching of documents based on classification tags, rather than on content.

To support category-based searching, CCL has been enhanced to include the search function: **Class()**. This function finds records that have been assigned category tags and can, of course, be combined within any Boolean expression as normal.

The **Class()** search function accepts a string as an argument and this string is interpreted as the name of one or more categories within the current scheme.

Note:

Each open database may be assigned a different scheme and that this indirect mapping is performed separately for each open database.

Once the names are matched, any category tags (record IDs) found are then located within the open database.

An example of the **Class()** function is shown below, being exercised on the imaginary classification database '*classtest*':

```
Search  Records   Command
1       276       BASe classtest
2       10        Find CLASS(financial) AND
australia
3       147       Find CLASS(economic)
4       181       Find CLASS(economic) OR
                    CLASS(financial) AND USA
```

Note:

*The **CLASS()** function does not produce specific hit locations within documents, thus a simple search for a category on its own (e.g. S=3, above) will not result in highlight points being shown in any report.*

For more in depth information on classification in TRIP, consult the following documentation:

- The **Define Class()** and **Find Class()** sections of the *CCL Command Reference*
- The document entitled, '*TRIP_White_paper_Classification.pdf*' which is located in the TRIPsystem installation 'doc' directory
- TRIPmanager help file (*TRIPmmc.chm*) which is located in the TRIPmanager installation directory, or accessible via the Microsoft Management Console's Help menu.

Fuzzy Searching

'Fuzzy' search implies searching for similarity, which can be very useful when searching for related terms, differently spelled or frequently misspelled words, or text which has many typographical errors. The CCL command modifier **FUZz()** (short form: **FUZ()**) is generally used in combination with the CCL **Display** command to generate a list of terms from which to choose. It uses this syntax:

```
d fuz(term) ↵
```

where *term* is the word of interest.

Examples:

Example 1:

```
d fuz(not) ↵
```

finds 163 occurrences of 'not' in database 'Alice' and two occurrences each of 'note' and 'knot':

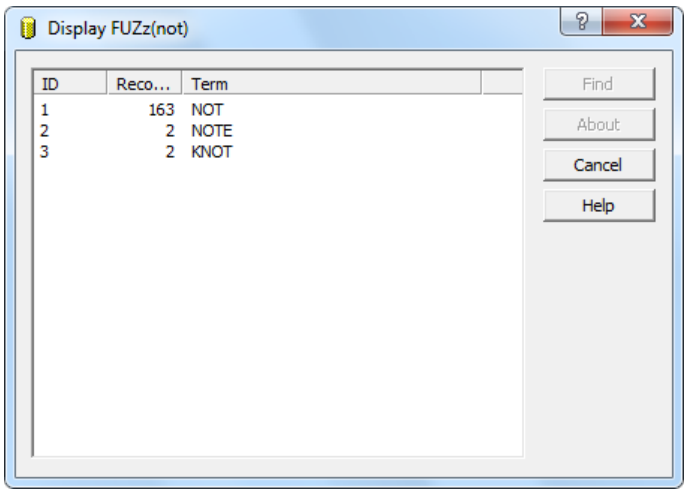

Figure 5–1 d fuz(neighbor) example

Example 2:

```
d fuz(alice) ↵
```

locates 427 occurrences of 'Alice', one of 'Alicee' and three of 'Slice':

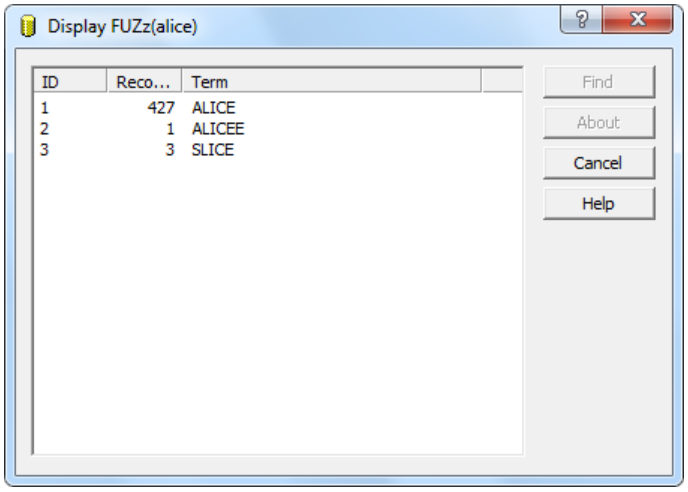

Figure 5–2 d fuz(alice) example

TRIP is attempting to locate terms which could be similar to the term given. This operation is by default based on an 'n-grams' algorithm. The term with the largest percentage similarity based on the 'edit difference' (in this case, 'Alice') receives the highest ranking and is placed at the top of the term list.

A search using **FUZz** may be further tailored by using three out of of four additional parameters, separated by commas:

- The first parameter is an integer dictating the percentage similarity, defined using edit distance, that any search result has to the search term. The range of permissible values for this parameter is from 1 to 100.

  Default Value: 75

- *The second parameter is depreciated and has no effect in normal fuzz searching; it can therefore be left out. However the leading comma must be used if any of the following parameters are entered, i.e.* **Display/Find FUZz(term,75,,2,1)**

- The third parameter sets the number of top ranked search candidates to include and *only* show an effect in a **FIND Fuz()** command.

  Default Value: 2

- The fourth parameter defines which fuzzy algorithm is to be used and can have one of two integer values:
  - 1 = n-grams
  - 2 = Sliding mask

  Default: 1

Note:

The default is also the recommended search algorithm.

If you wish, you can refer to the *CCL Command Reference* for more information on **FUZz()** command modifiers.

Continue with database **Alice** to work the next six examples:

```
Search  Records   Command
1       475       BASe alice
```

Example 3:

`d fuz(late,80,,,1) ↵`

Note:

In actual fact, as the second parameter is depreciated, the third parameter has no effect in a **Display** *command and the n-grams algorithm is the default, the above command could be entered thus:*

`d fuz(late,80) ↵`

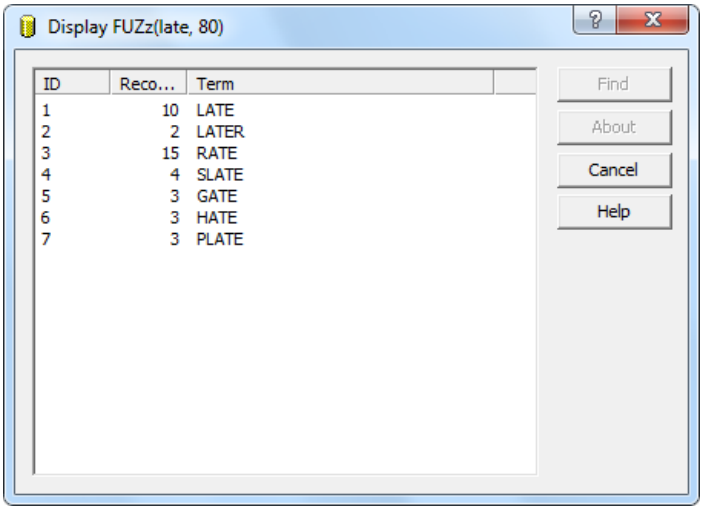

Figure 5–3 d fuz(late,80,,,1) example

This order displays expressions located using the 'n-grams' algorithm, which show an eighty percent similarity to the search term.

To illustrate how parameters affect fuzzy search performance, the terms and numbers of hits found for four sample parameter sets of the request **`Display FUZz(late...)↵`** from **Alice** are given below. As explained above, parameters that have no effect have been omitted:

Examples 4 through 8:

```
(...65,,,1)↵      (...75,,,1)↵ (...100,,,1)↵      (...6,,,2)↵
10   LATE         10   LATE    10   LATE          10   LATE
2    LATER        2    LATER                      153  LIKE
15   RATE         15   RATE                       55   LARGE
4    SLATE        4    SLATE                      43   LIVE
3    GATE         3    GATE                       21   LEAVE
3    HATE         3    HATE                       20   LIFE
3    PLATE        3    PLATE                      5    LOSE
3    ATE          3    ATE                        3    LACIE
2    LASTED                                       3    LINE
1    LATELY                                       3    LOVE
6    SLATES                                       1    LADLE
4    PLATES                                       1    LITH
```

Note:

The third parameter, omitted above, sets the number of top results used in a **Find** *command*

The **FUZz** modifier and its parameters can also be combined with **Find** and **DEfine** statements. These and the examples in the next section on 'Fuzzy Logic' are taken from database **Alice**:

```
bas alice ↵
Search  Records   Command
…
3       475       BASe alice
```

Example 9:

```
f fuz(late,80,,3) ↵
Search  Records   Command
…
4       27        Find FUZz(late,80,,3)
```

This order locates expressions which correspond to the top three results with an 80% similarity to the term 'late' (i.e. 'late', 'later' and 'rate'), located using the (default) 'n-grams' algorithm.

Example 10:

```
de fuz=80,1,7 ↵
'FUZZ parameters set to 80,1,7,1'
```

Note:

For technical reasons, unlike in **Find FUZz()** *or* **Display FUZz()** *commands, in a* **Define FUZz** *command, it is not possible to omit parameter number two, which must instead always be set to a value of '1'.*

The **DEfined FUZz** parameters above will apply to all subsequent searches until changed or this TRIP session ends. The **FUZz** 'late' example above now finds five additional hits:

Example 11:

```
f fuz(late) ↵
Search  Records   Command
```

```
…
5       40          Find FUZz(late)
```

It is not necessary to specify all three quantifiers in a **FUZz** parameter string. In addition to the search term, you may specify only the first:

Example 12:

```
f fuz(goat,95) ↵
Search  Records    Command
…
6       5          Find FUZz(goat, 95)
```

or the first and third:

Example 13:

```
f fuz(goat,95,,3) ↵
Search  Records    Command
…
7       14         Find FUZz(goat, 95, 3)
```

When using **Find FUZz()** or **Display FUZz()** It is possible to skip one or two parameters (using commas as place holders) and make a statement such as the following one, which uses only parameter number three (the omitted parameters will remain a previously defined):

Example 14:

```
f fuz(goat,,2) ↵
```

Truncation or masking characters *cannot* be used in the **FUZz()** search term, nor are these necessary given the search capabilities of this feature.

Fuzzy Logic and Relevance Ranking

Notes:

> *This section refers to the CCL **FUZz** command which, despite it's rather confusing name, has no connection at all with the commands mentioned in the preceding section; i.e. **Define FUZz()**, **Display FUZz()** and **Find FUZz()**.*

Using **FUZz** as a **Find** command results in a search where the Boolean conditions are treated less rigidly than usual.

With **FUZz**ed searches, the highest-ranking hit records are those which *best satisfy* the original search condition. Each record's ranking in the search result is determined both by the number of matching terms it possesses and by the distance between them—the shorter the distance, the higher the rank.

For example,

Example 15:

```
fuz mome raths jabberwock borogoves ↵
Search  Records    Command
…
8       2          FUZz mome raths jabberwock
borogoves
```

finds one record (number 372) containing one occurrence each of 'jabberwock', 'borogoves', 'mome' and 'raths' and one record (number 248) with three occurrences of 'jabberwock' and two each of 'borogoves', 'mome' and 'raths'.

Field **WEIght**s (short form: **WEI**) may be assigned to certain fields to make hits in those fields more significant than others. Valid field weights range from one (the default value) up to an undefined positive integer.

Assuming a starting field **WEIght** of one, this example assigns the **person** and **txt2** fields eight times the importance of other fields with a **DEfine** request:

Example 16:

```
de wei(person,txt2)=8 ↵
```

When **Show**ing records from a **FUZz**y search, you should use the modifier **SORt=FRequency** to **SORt** them on the ranking. Using **SORt=FRequency** on a non-**FUZz**ed search result causes the records to be sorted on the number of occurrences in each record.

Using the 'mome/rath/jabberwock/borogove' example above, type a **Show SORt=FRequency** order (short form: **S SOR**=**FR**):

Example 17:

```
s sor=fr ↵
```

Since **WEIght** has been predefined as **(person,txt2)=8**, the records are displayed in reverse (numbers 248 and 372), as record 248 contains more hit terms than the other records.

Searching More Than One Database

When a database is opened for the first time during a search session, some checking and reading is done that is not performed again if the database is reopened in a subsequent **BASe** order. Among other things, the system checks that the user has reading rights to the database and reads the information that forms the **STatus** list.

When a database is opened, an initial search is made which finds all the records in the database before it was last indexed (any records added since the last index will not be found). These are shown as the first search in the search history.

To begin, open databases **Alice** and **Corr** together using the **BASe** order:

```
bas two_databases=alice,corr ↵
Search  Records   Command
…
9       574       BASe two_databases=alice, corr
```

Two_Databases is a *multifile*, a temporary name (up to sixteen characters in length) that the user gives to a set of databases. A multifile can be considered a database in its own right, which has been created to hold all the records of the individual databases mentioned in the **BASe** order.

During the current search session, multifile names will appear in the Open Database window as if they were permanent databases. A maximum of 250 databases may be open at the same time during a search session.

Note:

Do not confuse a multifile, which is temporary, with a cluster, a permanent collection of databases which are always opened together.

If you open several databases and issue a **Show** or **Print** order, the records are shown in the same sequence as that in which the databases were initially opened. A **Show** order for a search that hits records of both databases outputs the records of **Alice** first.

Suppose that later during the same session you wish to open the demonstration database **Thesali** at the same time as the other two. You can do this using one of these orders:

```
bas three_databases=thesali,two_databases ↵
Search  Records   Command
…
10      713       BASe
three_databases=thesali,two_databases
```

or

```
bas three_databases=thesali,alice,corr ↵
Search  Records   Command
…
11      713       BASe
three_databases=thesali,alice,corr
```

TRIP informs us that there are now 713 records in the multifile. This is diagrammed in the figure below, in which two multifiles with records in common are opened (Figure 5 – 4):

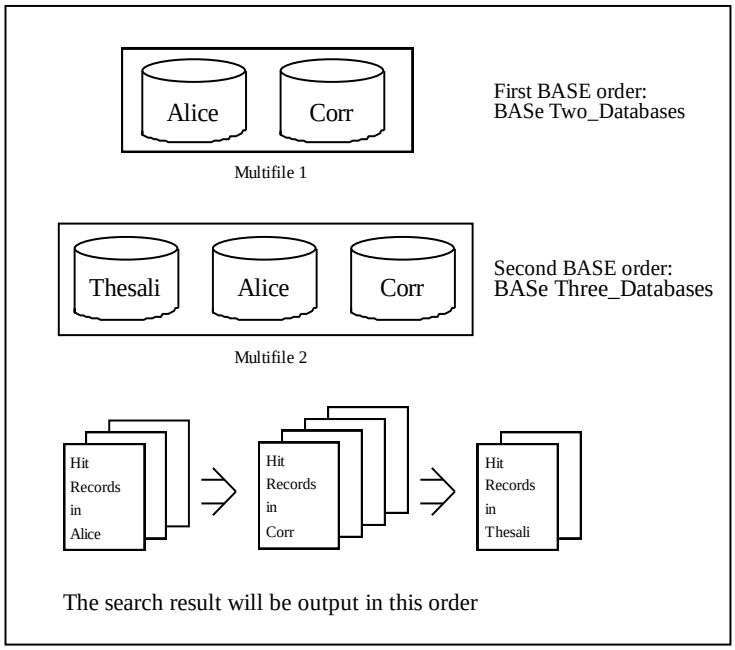

Figure 5–4 The record sequence when opening two multifiles

A **Show** order for a search that finds records in all the databases now outputs the records in the following order: **Alice**, **Corr**, and **Thesali**, as that is the order in which the databases were opened.

Try this search:

```
f send and all ↵
Search  Records   Command
```

```
…
12      7          Find send AND all
```

When you **Show** this result, you will see that TRIP found records containing both 'send' and 'all' in only two of the three databases open, with this output:

from **Alice,**

Record 135
Record 360
Record 386

and from **Corr**,

Record 52
Record 92
Record 94
Record 97

As **Alice** was the first database opened, the records from **Alice** will be displayed first in standard (ascending) numerical order (record numbers 135, 360 and 386). These will be followed by the records found in **Corr** (numbers 52, 92, 94 and 97).

Using the **REVerse** modifier in the **Show** order will reverse both the display sequence of the databases and the order of the records in each database.

Try this now:

s rev ↵

The database order is now **Corr**-**Alice**, and the hit records are now displayed in descending numerical order within each database:

from **Corr,**

Record 97
Record 94
Record 92
Record 52

and from **Alice**,

Record 386
Record 360
Record 135

If the **SORt** modifier is used in an output order with the intention of having the hit records of all databases sorted, the qualifier **MERGe** must be included. This ensures that the hit records from all of the databases are first 'merged' and *then* sorted. If **MERGe** is omitted, the records are sorted within each database, and then the sorted records for each database are output in the order in which the databases were opened:

s sor=rname,person merg ↵

Note:

*If merged sorted outputs are always required within a session, this can be made the default by using **DEfine MERGe**:*

de merg ↵

TRIP confirms your request with the message "Sort merge of databases."

Searching One of Several Open Databases

If you must search simultaneously in several open databases containing the same field names, you can restrict your searches to only one of the databases by appending the field name you wish to search to the corresponding database name as a suffix.

For example, to search the multifile sub-database **Corr** for all references to letters sent or received, you might search the body or text of each letter like this:

```
f corr.content=letter ↵
Search  Records   Command
…
13      27        Find corr.CONTENT=letter
```

History explains that twenty-seven records have been found that contain 'letter' in the **content** field of database **Corr**.

DEfining a SCope Limit

TRIP allows you to define a given set of records within a database or cluster of databases, and limit all subsequent orders to only these records. This saves having to continually **AND** the search result with the one that defined the given set, and allows unrestrained text searching in a set of records that may have been defined by a non-textual search (for example, when using a record number range).

If you have executed all of the sample searches given so far, your History window will contain the following:

```
Search  Records   Command
1       475       BASe alice
2       99        BASe corr
3       475       BASe alice
4       85        Find FUZz(mouse, 80, 3)
5       90        Find FUZz(mouse)
6       14        Find FUZz(goat)
7       14        Find FUZz(goat, 95, 3)
8       3         FUZz mome AND rath AND jabberwock AND
                      Borogove
9       574       BASe two_databases=alice, corr
10      713       BASe three_databases=thesali,
                      two_databases
11      713       BASe three_databases=thesali, alice, corr
12      7         Find send AND all
13      27        Find corr.CONTENT=letter
```

To limit the number of records available for later orders to the result set of a particular search, combine the **DEfine** and **SCope** (short form: **DE SC**) statements with the search order number:

```
de sc s=12 ↵
Search  Records   Command
…
14      7         DEfine SCope S=12
```

This example limits subsequent searches to the seven records (S=12) which contain the terms ‘send’ and ‘all’ in the databases **Thesali**, **Alice** and **Corr**.

If the search set you would like to use resulted from the last search made, the order to limit the scope of subsequent orders to that results is:

```
de sc s=0 ↵
Search  Records   Command
…
15      7         DEfine SCope S=14
```

In this example, S=15 duplicates the scoping of S=14. This order can be used repeatedly to limit the search result set after any successful search.

Remove the scope limit now so that we once again can search all 714 records of **Three_Databases** by using **Define** and **SCope**:

```
de sc ↵
```

TRIP announces “Current scope deleted”.

When you need only one or two **Find** or **Display** orders to operate on a small set of records, rather than

- defining the scope,
- giving the order,
- and switching the scope off again,

it may be faster and easier simply to refer to the search that defined the limited result set. Whether the search set is the first successful search executed, the last or somewhere in between, simply add the search number to the command you wish to run.

The following **Find** order looks for the term ‘mail’ within a set of twenty-seven records containing the term ‘letter’ in field **content** of database **Corr** (S=13 in History):

```
f mail and s=13 ↵
Search  Records   Command
…
16      1         Find mail AND S=13
```

A request to **Display** the same information looks like this:

```
d s=13 content=mail ↵
```

or

```
d s=13 mail ↵
```

Another order which limits the scope of orders, **DEfine SCope SDI**, is covered later in this chapter.

Other Forms of the AND Operator

On its own, the **AND** operator simply adds one search condition to another, to find all records which satisfy the conditions written before **AND** after. To be more precise and specify that the conditions are to be met within a certain region within a record, use these symbols:

Operator	Function
AND.F	hits must occur within the same field
AND.P	hits must occur within the same paragraph
AND.S	hits must occur within the same sentence
AND.W	hits must occur within the same word

Table 5–1 AND Operator forms

These operators may be used in combination, although parentheses are necessary wherever ambiguities might arise.

Try these **Find** examples from **Three_Databases**:

Example 1:

```
f s=11 and crea# and.w #ti# ↵
Search  Records   Command
…
17      2         Find S=11 AND crea# AND.W #ti#
```

finds records containing the ‘crea’ and ‘ti’ truncations in the same word (that is, ‘creation’ and ‘creating’, not ‘cream’, ‘create’, ‘created’, ‘creature’ or ‘creatures’).

Example 2:

```
f creating and.f creation and s=11 ↵
Search  Records   Command
…
18      1         Find creating AND.F creation AND
S=11
```

finds records containing ‘creating’ and ‘creation’ in the same field.

Example 3:

```
f s=11 and ancient and.s history ↵
Search  Records   Command
…
19      2         Find S=11 AND ancient AND.S
history
```

finds ‘ancient’ and ‘history’ in the same sentence.

Example 4:

```
f s=11 and ancient and.p mystery ↵
Search  Records   Command
…
20      1         Find S=11 AND ancient AND.P
mystery
```

finds ‘ancient’ and ‘mystery’ in the same paragraph.

Changing Format during a Show Order

It is possible to change reports while browsing through the results of a search. This is useful where different formats are available for the database, and different types of formats are appropriate depending on the contents of the record.

For these examples, re-examine the results of search number twelve from the History window (**f send and all**) with the command:

```
s s=12 ↵
```

Record number 135 from **Alice** is the first to be displayed.

Move forward one screen. You should see the last eight lines of text from record 135 ('…when she had got…' to '…a very difficult game indeed.') and the first four lines of text from record 360 ('What tremendously easy riddles…' to '…that Alice could hardly…').

The current record is always the one displayed at the bottom of the screen, in this case number 360. To view only the current record (now number 360) during a **Show** order, use its modifier **Record** (short form: **R**):

```
s r ↵
```

To see the same record using a different format, you can request the new arrangement by adding a **Format** command. This illustration uses a fictional format called **New_Format**:

```
s f=new_format r ↵
```

or

```
s r f=new_format ↵
```

To try this using the previous search S=12, escape from the current record using **Continue** ↵ (short form: **C**), and scroll down one screen further to record 52 of **Corr**. Try the above example using Format 2:

```
s f=2 r ↵
```

TRIP shows only the date and the senders' and receivers' names, addresses and countries, as defined in **Corr**'s Format 2.

Note:

*There are a number of ways to 'escape' from a **Show Record** command:*

- *To make a new search, type a new **Find** order.*
- *To return to the top of your original search result, enter any new **Show** request.*
- *To return to the record following that original record, use **Continue**.*

Try an example:

`s s=12 ↵` (shows search result 12)

Scroll to record 360

`s r ↵` (show that record)

`c ↵` (continue viewing)

You are returned to record 386, which follows record 360

Field names can also be used in place of the format name in a **Show Record** request. Using the last search result obtained
(S=20 Find … ancient and.p mystery), ask to see only the contents of the **person** and **speaker** fields for that record:

```
s f=person,speaker r ↵
```

This does not affect an original **Show** order, which you can view at any time using **Continue**.

To change the format for the current record and all following records in the search result, use **FRom** in the search order. Using the hypothetical format **New_Format**:

```
s f=new_format fr r ↵
```

This order is equivalent to a new **Show** order on the same search set, and shows only the records from the current record onwards. Using S=12 as an example once again, **Show** that search result:

```
s s=12 ↵
```

and scroll to record fifty-two, the first hit from **Corr**.

Printing

Print Commands

There are four **Print** (short form: **P**) commands in TRIP, which affect the immediacy with which the print job is started. They are:

```
p HOLD ↵
p NO HOLD ↵
p NOW ↵
p WAIT ↵
```

Print HOLD is the default, the command used by TRIP when a **Print** order is given without a modifier. It collects all of the print requests made during a session, and submits them as a batch job when the TRIP session is ended.

Advantages

Having **Print HOLD** as the default **Print** command is useful, because all **Print** requests held can be listed using the command **Print?** (short form: **P?**), which shows both the print order number and the command used to generate each print job.

When necessary, you can **DELete** a print request that has been generated with **Print HOLD** using the command

```
del p=n ↵
```

where *n* is the print request number you wish to delete.

You can also delete all waiting **Print HOLD** jobs using

```
del p=all ↵
```

If you have no **Print HOLD** jobs in the queue and attempt to delete one or more, TRIP informs you that it has "No PRINT HOLD orders stored."

Disadvantages

The search which generated the result set to be printed must not be deleted. If you must delete a search after placing a **Print** order, one of the other **Print** commands should be used. If you attempt to delete a search after instructing TRIP to print it, you will receive the message "All requested sets not deleted due to pending PRINT order."

Print NO HOLD, as the name suggests, does not hold print requests until the end of the session, but instead submits them to the batch queue as soon as the order is given.

Print NOW and **Print WAIT** are different, in that the print job is run as a sub-process rather than as a separate process in a batch queue. The advantage of this approach is that the print job cannot be held up by other jobs in the queue.

Print NOW runs a sub-process in parallel to the TRIP session, and so allows searching to continue.

Note:

If the host session is terminated before the sub-process completes, the print job will fail an if the sub-process is still running when the TRIP session is terminated, TRIP issues a warning to this effect.

Print WAIT runs a sub-process, and blocks all further activity until the print job has completed. This is useful where all existing search sets are to be deleted, to allow searching from scratch.

Other Print Commands

To **Print** the results of a search, add the search result number to the command:

```
p s=12 ↵
```

If you have more than one database open at a time (such as if you had used the order **bas two_databases=alice, corr**), you can print information from only one of them with the **Print BASe** (short form: **P BAS**) order:

```
p bas r=corr ↵
```

and to print the contents of an entire (open) database, use

```
p bas ↵
```

Printing Locally

If you have a locally-attached printer and would like to print a hard copy reference of something displayed on your screen (**Display**, **STatus**, or **Help** screens, etc.), use the **Print Local** (short form: **P L**) command:

```
p l ↵
```

Note:

This does not enable you to make a screen shot of the records contained in a search result if you have only the History window on screen! If you are not displaying the records in a Show window, you will get only a snapshot of the History screen.

To **Print** the results of a search locally, add the search result number to the command:

```
p l s=12 ↵
```

If you have more than one database open at a time (such as if you had used the order **bas two_databases=alice, corr**), you can print information from only one of them with the **Print BASe** (short form: **P BAS**) order:

```
p l bas r=corr ↵
```

It is also possible to define the page size or override a page size limitation when printing on a local printer. Use **DEfine PAge** (short form: **DE PA**), which has two parameters, **rows** and **columns**, and an additional instruction, **Form Feed** (short form: **FF**) to direct the printer to make a form feed at the end of the output file. For example:

```
de pa=60,80 ↵
```

```
"Page settings for local printer redefined to 60,
80"
```

defines a page size of sixty rows and eighty columns,

```
de pa=60,80,ff ↵
"Page settings for local printer redefined to 60,
80, ff"
```

defines a page size of sixty rows by eighty columns and a form feed at the end of the print job, and

```
de pa ↵
"Paged output"
```

resets the printer settings to their defaults.

Dates, Times, Numbers and Integers

Numeric range searching can be performed on all of the above-mentioned types of fields, and works in the same way for each. We will consider all of the searches that are possible with these data types, using the integer field in **Alice** called **chaptnr**.

Delete the contents of Search History with **DELete S=All**.

Open **Alice** (S=1) and try these **TO** and **FRom** examples (shown with TRIP's messages of acknowledgment) for:

…searching up to and including a value:

```
f chaptnr=to 2 ↵
Search  Records   Command
1       475       BASe alice
2       81        Find chaptnr=TO 2
```

…from a value inclusive onwards:

```
f chaptnr=fr 11 ↵
Search  Records   Command
…
3       46        Find chaptnr=FRom 11
```

…an exact value:

```
f chaptnr=1 ↵
Search  Records   Command
…
4       39        Find chaptnr=1
```

…a closed range of values:

```
f chaptnr=1 to 3 ↵
Search  Records   Command
…
5       124       Find chaptnr=1 TO 3
```

…and any combination of these coupled with any other kind of content search:

```
f chaptnr=to 1,fr 11 and rabbit ↵
```

```
Search  Records    Command
…
6       30         Find (chaptnr=TO 1, FRom 11) AND
rabbit
```

The effect of **FRom**, **TO** and the closed range are illustrated below.

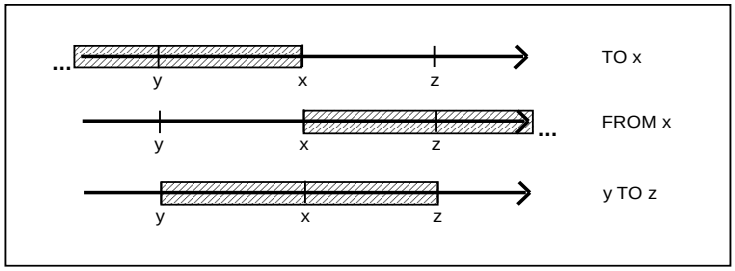

Figure 5–2 FRom, TO and the Closed Range

Searching of this kind can be performed on any field of type **DAte**, **TIme**, **NUmber** and **INteger**. Some of the search possibilities for a hypothetical field of type **NUmber** called **Number_Example** are shown below:

…up to and including a value:

```
f number_example=to -2.5 ↵
```

…from a value inclusive onwards:

```
f number_example=fr 6.5 ↵
```

…an exact value:

```
f number_example=5.5 ↵
```

…a closed range of values:

```
f number_example=-1.5 to 3.5 ↵
```

…and any combination:

```
f number_example=to -2.5,-1.5 to 3.5,fr 6.5 and
entropy ↵
```

Searching by Minutes and Seconds

The usual format of a field of type **TIme** is **hh:mm:ss**, where **hh** represents the hours in twenty four hour clock, **mm** the minutes, and **ss**, seconds. This is referred to as **TIMEForm**=1 (short form: **TIMEF**).

As long as colons are used as delimiters, leading zeros can be ignored, and if you keep in mind that *hours* are the most significant values, then trailing seconds (and, if desired, trailing minutes) can also be ignored.

These rules allow TRIP to interpret the following examples using the fictional **TIme** fields **start_time**, **leave_time**, **lunch_time**, **duration** and **lap_time** without contention:

```
f start_time=8:5 ↵              (8:05:00AM)
f leave_time<18 ↵               (before 6:00:00PM)
f lunch_time=12 to 13:50 ↵      (12:00AM to 1:50PM
inclusive)
```

This format (**hh:mm:ss**) works well if fields of type **TIme** always contain full time values. If they happen to contain only minutes and seconds, however, all values are then of the form **00:mm:ss** and would require search orders like this one:

```
f duration=00:10:00 to 00:12:30 ↵
```

to locate values for **duration** between ten and twelve and one-half minutes.

As this is rather cumbersome, you may wish to make the minutes the most significant value using **TIMEForm=2**. In this way you can search for minute and second values without entering leading zeros and colons in your search orders:

```
f duration<25:59 ↵              (25 mins, 59 secs)
f lap_time=3 to 4 ↵             (3 to 4 mins
inclusive)
```

and instruct TRIP to interpret the first value as a minutes value. This will not prevent you from entering a full-format time value such as **1:59:59**, e.g.:

```
f lap_time=59 to 1:59:30 ↵      (59 mins to 1 hr 59
mins                                        30
secs inclusive)
```

To change the default **TIMEForm**, use the following **DEfine** order:

```
de TIMEForm=n ↵
```

n being a unit of time. Where *n=1*, the hours value is the most significant, where *n=2*, the minutes are the most significant, and where *n=3* the seconds are the most significant. TRIP recognizes a request such as this:

```
de TIMEF=3 ↵
```

with the message “TIMEFORM now set to 3”.

MEasure and FRequency

MEasure

The command **MEasure** (short form: **ME**) is applicable to fields of type **NUmber** and **INteger** in a search result. It generates common statistical measures such as average and standard deviation, describing the data in a single **NUmber** or **INteger** field in the records of a search result and presenting them in a table. If a value contains too many digits, it is replaced by the overflow symbol ########.

The order

me chaptnr ↵

or

me s=0 chaptnr ↵

(where **chaptnr** can be of type **NUmber** or **INteger**) displays the field name being analyzed (**chaptnr**), the database name to which the field belongs (**Alice**) and the search result number under consideration (Search: 6), in addition to statistical information for the latest search result. The order:

me s=5 chaptnr ↵

provides a statistical summary for the records in S=5.

FRequency

The command **FRequency** (short form: **FR**) gives information on the distribution of values within a given field in the records of a search result, and works with fields of any type. This is done by orders such as:

fr field1 ↵

or

fr s=3 *field2* ↵

where *field1* and *field2* are the target fieldnames.

For example

bas alice ↵

```
Search  Records   Command
…
7       475       BASe alice
```

fr S=0 person ↵ (or just `fr person`)

shows the frequency of occurrence of the **persons** in the **Alice** database, and

fr txt2 ↵

shows the frequency of individual words in the **txt2** field.

Both a range and a step within the range may be specified for fields of type **NUmber** and **INteger**. For example, if the imaginary field **age** contains peoples' ages, and only people old enough to work but not yet of retirement age are being considered, a frequency analysis of the ages fifteen to sixty-five would be appropriate:

fr age 15 to 65 ↵

In order to have these summarized into five groups (intervals or bands) of ten years span, the *step modifier* ten is added:

```
fr age 15 to 65 10 ↵
```

Try this example from **Alice**, which examines the frequency of values between three and twelve found in the field **chaptnr**:

```
fr chaptnr 1 to 15 ↵
```

TRIP provides the field name being analyzed, the database to which it belongs, the search being examined and the number of entries found, in a similar fashion to **MEasure**.

Each figure for **chaptnr** appears in the Value column as '1-1', '2-2' and so on, where '1', '2' etc. represent both the lowest and the highest value within the interval specified. Since we have not requested that the output be divided into intervals or grouped in any way, each value for **chaptnr** appears on a separate line, making the *default interval* or *step* equal to one. If we are interested in looking at the frequency of chapter numbers six through twelve and would like them to be grouped into pairs or sets of two, we could combine **chaptnr** with *step modifier=2* in this way:

```
fr chaptnr 6 to 12 2 ↵
```

TRIP informs us that **chaptnr**s with values up to five represent 210 (44.2%) of the records found, while values six or seven are present in 95 (20.0%) of records, eight to nine, 106 records (22.3%), ten to eleven, 38 records (8.0%) and twelve, 26 records (5.5%).

Timestamp

Each record in a TRIP database is time stamped with the date and time the record was first created, and thereafter when it was last modified. This information is stored in field **TStamp** (short form: **TS**) and may be used in searches similar to these from **Corr**:

```
bas corr ↵
Search  Records   Command
…
8       99        BASe corr
f ts=1992 ↵
Search  Records   Command
…
9       92        Find TStamp=1992
f ts=92 ↵
Search  Records   Command
…
10      92        Find TStamp=1992
f ts=1992-01-01 00:00:00 to 1992-12-31 23:59:59 ↵
Search  Records   Command
…
11      92        Find TStamp=1992-01-01 0:00:00 TO
                      1992-12-31 23:59:59
```

The three orders all produce the same search result, finding all the records in the open database that were created or modified in 1992.

Timestamp searching works in the same way as date and time searching, i.e. lists of values and/or intervals may be used:

```
f ts=to 1992-02-01 12:00,1992-06-01 13:30:30 to
1993 ↵
Search  Records   Command
…
12      99        Find TStamp=TO 1992-02-01
12:00:00,
                      1992-06-01 TO 1993
```

In a timestamp order, a time must not be entered on its own, i.e. without a date to its left, or TRIP will produce the message "*n o'clock* is no legal timestamp date" (*n o'clock* being the time value in question).

You can also combine **TStamp** searches with other search parameters using logical operators. To find all of the records in **Corr** which were entered or modified after February 1, 1992 and which contain the term 'sven' in field **sname**, type

```
f ts=fr 1992-02-01 and sname=sven# ↵
Search  Records   Command
…
13      4         Find (TStamp=FRom 1992-02-01) AND
                      sname=sven#
```

Timestamp SDI

SDI stands for **S**elective **D**issemination of **I**nformation. This TRIP feature uses Timestamp information in such a way that users can view only those records which have been modified or added to a database since their last TRIP session.

Some form of control is required, and this is implemented using **SDI** orders. At the start of a normal SDI search session, give the **DEfine SCope** order:

```
de sc sdi ↵
Search  Records   Command
…
14      9         DEfine SCope SDI
```

to confine your searching to only those records which have been added or modified since you last updated the **SDI** field.

If at the end of the viewing session you give the **UPDate** (short form: **UPD**) order:

```
upd sdi ↵
```

TRIP will acknowledge your request with "SDI values updated.", and the current date and time will be written in the SDI Timestamp marker in your User Profile for that database. Any records which are modified or added to the database will have a Timestamp later than this SDI Timestamp value. However, the SDI Timestamp does not *have* to be updated at the end of a session. If it is not, you will be able to view all of the old records as well as recently modified or added records the next time you give the **DEfine SCope SDI** order.

Conversely, the SDI Timestamp can be explicitly updated to any value. If, for example, you wanted to see any records that have been modified or added since the beginning of the month, the order in this case would take the form:

```
upd sdi databasename ts=1992-07-01 ↵
```

TRIP holds separate SDI Timestamps for each database and user.

Using SDI has no effect on normal searching. If no **DEfine SCope SDI** order is given, searching takes place in the usual way. If you have used a **DE SC SDI** order, the **SCope** can be removed using **DEfine SCope** without modifiers or qualifiers, regardless of whether SDI is updated or not.

Record Number SDI

TRIP stores the SDI Record Number in the same way as the SDI Timestamp marker, and can be used for SDI search sessions as well.

The essential difference is that whether a record is added or modified, its Timestamp value is updated, but a record is only allocated a new record number if it is added. Hence SDI searching on record numbers relates only to newly added records, whereas SDI searching on Timestamps relates both to newly added and newly modified records.

In all other respects, both SDI variants work in the same way. Both SDI values and the SDI Record Number can be updated using the **UPDate SDI** order:

```
upd sdi databasename r=n ↵
```

You will receive the message “SDI values updated.” once again.

The only difference is that when defining the scope to include only newly added records rather than those that have been updated, the **DEfine SCope SDI** order is given the modifier **NO UPDate** (short form: **NO UPD**) as follows:

```
de sc sdi no upd ↵
Search  Records   Command
n       h         DEfine SCope SDI NO UPdate
```

where *n* represents the search number and *h* the number of hits.

Searching Tupled Fields

TRIP records may be used to hold tables of numbers or terms, similar to a spreadsheet. The columns are held in fields, and the rows in linked levels of subfields. The values in each row are then referred to as a *data tuple*.

The subfields of a database are not physically tupled in the database, but the link is maintained on data entry and in **Show** orders. In order to maintain the tuple when searching, the operators **AND.T** and **NOT.T** are necessary.

This concept is most easily demonstrated with an example:

In the hypothetical database **Holidays**, for each resort facility there is a record for each week of the season in the province. The Tourist Office records the hours of sunshine, the millimetres of rainfall and a forecast of ‘sun’, ‘cloudy’ or ‘rain’ for each day as tupled fields. In the first subfield of each of these three fields are the figures for Monday, the second subfields hold the figures for Tuesday, the third for Wednesday and so on.

Each record contains the resort name and week number, and seven subfields for each of the three fields.

A 'bad week' is one which contains one or more days where the sun shone for less than five hours, or where the rainfall was more than 2mm in any one day. Bad weeks can be found using the order:

```
f sun < 5 and.t rain > 2 ↵
```

A week which has an inaccurate rain forecast is one which has any one day for which the forecast was rain, but it does not. These are found using:

```
f forecast=rain and.t rain=0 ↵
```

A week which has an erroneous sun forecast is one for which sun or clouds were predicted, but which experienced rain. These are found using:

```
f rain>0 not.t forecast=rain ↵
```

All of these conditions find matches for any given day, and the orders find records which have any days that match. They indicate for which resorts in which weeks the statements are true.

Non-tupled searching is still available, as tupling occurs in data entry, searching and output, and is not inherent in the database structure.

For example, the order:

```
f rain > 0 not forecast=rain ↵
```

finds any week in which a daily rainfall was recorded, but there was no daily forecast for rain. However, without the qualification of being in the same tuple, it is not known whether or not this was the forecast for that day only.

As **TExt** fields have no subfields, fields of any other type can be tupled. Field names must always be used when searching in tupled fields.

Defining Maximums and Minimums

Maximums

Upper limits or **MAXimum**s (short form: **MAX**) may be set to the number of hits of a **Find** order, the number of terms of a **Display** order, or the number of records to be **SORt**ed or **Print**ed. To view the current values of some of the definable system parameters, use:

```
de? ↵
```

To raise or lower the **MAXimum** number allowed, use one or more of these **DEfine** orders:

DEfine Order	Default Max	TRIP's Confirming Message
`de f max=10000 ↵`	Unlimited	"New limit for FIND set to 10000."
`de d max=2000 ↵`	1,000	"New limit for DISPLAY set to 2000."
`de sor max=500 ↵`	1,000	Interactively: "New limit for SORT set to 500." (Unlimited during printing)
`de p max=100 ↵`	Unlimited	"New limit for PRINT set to 100."

```
de map max=300 ↵   1,000          "New limit for MAP set to
                                        300."
```

Figure 5–5 Examples of DEfine MAXimum orders

These limitations save resources when a user gives an unusually long or complex order that might take a long time to process. If any of the limits are reached, TRIP provides a message like "More than *n* term hits, be more specific please." and stops processing the order, which can then be edited to produce a more modest result.

However, the limit for **Print** orders is ignored when a record list is specified, as follows:

```
p r=fr 1 ↵
```

or

```
p bas=alice r=fr 1 ↵
```

Also **Print** orders from the control database such as:

```
p bas ↵
p user ↵
```

are unaffected by the limit.

Minimums

There may be an occasion where a search result of zero hits is useful, for example, when saving a search order for future use. The default for a **MINimum** (short form: **MIN**) search result is one, so that 'No hits' results only in a message, and the result is not added to the search history.

To set the minimum number of hits to zero, use the command:

```
de f min=0 ↵
```

The normal default (one) can be reset by giving the following:

```
de f min=1 ↵
```

Note:

For more information on **Define MAXimum/MINimum**, *see the* CCL Command Reference *section for the command.*

Combinations of Fields: Views

A **VIew** (short form: **VI**) is a group of fields, and may be defined using either field names of type **PHrase** and/or **TExt** or the type names themselves (i.e. **PHrase** or **TExt**). A **VIew** may be given a name, in which case it can be used in place of a field name or a field type, or it can simply be made the default for **Find** orders.

For the **Corr** database, a **VIew** could be defined to include both sender and recipient names, and another view specified to include both the sender and recipient countries.

This is achieved with these orders:

```
de vi see_names=sname,rname ↵
Search  Records   Command

…
15      0         DEfine VIew see_names=sname, rname
```

```
de vi see_countries=scountry,rcountry ↵
Search  Records   Command
…
16      0         DEfine VIew
see_countries=scountry, rcountry
```

The following search orders are now valid:

```
f see_names=sven# and see_countries=sweden ↵
Search  Records   Command
…
17      1         Find see_names=sven# and
see_countries=sweden
```

```
d see_countries=# ↵
(Display see_countries=#, 24 terms)
```

```
d see_names=paul ↵
(Display see_names=paul, 3 terms)
```

The **VIew** name is used in the same manner as a field name, so it is possible to include it in the definition of another **VIew**, as seen below:

```
de vi see_all_names=scomp,rcomp,see_names ↵
Search  Records   Command
…
18      0         DEfine VIew see_all_names=scomp,
rcomp,
                     see_names
```

Just as it is possible to use field names in a **VIew**, it is also possible to use field types. If no field is specified in a search order, TRIP searches by default in all fields that are of type **TExt** or **PHrase**. This default can be changed using a **DEfine VIew** order such as:

```
de vi=see_all_names,te ↵
"Default VIEW redefined."
```

Searching will occur only in the four fields defined by the **VIew** See_All_Names, as well as in all **TExt** fields. No **VIew** name is used in this kind of command.

Field names and types and **VIew** names can be used in combination in both types of **DEfine VIew** order.

To restore the normal default, use this order:

```
de vi=te,ph ↵
"Default VIEW redefined."
```

Use the **DEfine?** order to look at any **VIew**s which may currently be defined and their related fields.

Display Features

Display of the Contents of a Field

The orders:

```
d sname=# ↵
(Display sname=#, 38 terms)

d see_all_names=# ↵
(Display see_all_names=#, 103 terms)
```

where **sname** is the name of a **TExt** or **PHrase** field and See_All_Names is the name of a **VIew**, displays a list of all the terms in that field or view provided that their number does not exceed the current limit set by **Display MAXimum**.

Note:

*This syntax corresponds to **Find** orders looking for non-empty fields.*

Display of the Terms of a Search Result

The orders:

```
de vi see_all_names=rname,sname,rcomp,scomp ↵
"VIEW SEE_ALL_NAMES redefined.'

f #son ↵
Search  Records   Command
19      21        Find #son

d s=0 #son ↵
(Display S=0 #son, 12 terms)
```

displays a list of all the terms ending with 'son' in the records of the nineteenth search result.

If the terms 'Larsson' and 'Johansson' were selected from the list, the new search result would correspond to the order:

```
Search  Records   Command
…
20      6         Find S=19 AND.R ("JOHANSSON" OR
                      "LARSSON")
```

TRIP automatically alphabetizes your selections in the CCL statement.

Display Sorted by Frequency

To see the list of terms sorted in descending order of frequency instead of alphabetically, the modifier **SORt=FRequency** is used in the **Display** order, as follows:

```
d sor=fr see_all_names=#son ↵
(Display SORt=FRequency name=#son, 7 terms)
```

The term occurring most frequently is 'Rolf Larsson', which heads the list. Terms with the same frequency appear alphabetically by frequency, hence the three terms with two occurrences each are ordered by first name (*Helen* Hasselberg Nilsson, *Orvar* Svensson and *Rolf* Larsson), as are the three terms with one occurrence each (*Erik* Andersson, *Ferdinand* Peterson and *Steve* Jackson).

As many terms as may be displayed may be sorted on frequency (see the section on 'Fuzzy Searching' at the beginning of this chapter for more information on **FUZzy Display**).

Defining the ',' and ' ' to Represent Other Operators

The Logical OR Operator

As has been shown, the logical **OR operator** allows alternative search conditions to be specified. In the following order:

```
f london or boston or new york ↵
Search  Records   Command
…
21      13        Find london OR boston OR new york
```

if a record satisfies any of the conditions, it is considered a hit.

The comma character may be used to represent the **OR operator** using the **DEfine** order:

```
de ,=or ↵
"Comma delimiter is now treated as OR"
```

This allows an order of the form:

```
f london,boston,
new york,berlin,stockholm,san francisco ↵
Search  Records   Command
…
22      98        Find (london, boston, new york,
berlin,
                      stockholm, san Francisco)
```

The Logical AND Operators

The comma character could be defined to represent the logical operators **OR** or **AND.*x*** with these orders:

```
de ,=or ↵
"Comma delimiter is now treated as OR"

de ,=and.s ↵
"Comma delimiter is now treated as AND.S"

de ,=and.f ↵
"Comma delimiter is now treated as AND.F"
```

The latter order would make the following two requests equivalent:

```
f jan, sven# ↵
Search  Records   Command
…
23      3         Find jan, sven#

f jan and.f sven# ↵
```

```
Search  Records   Command
…
24      3         Find jan AND.F sven#
```

The **SPace** character (short form: **SP**) can also be used to represent the **AND.*x*** operator (where *x* represents the operator suffix), and it goes one step further than the comma character in that it allows the definition to apply to specific fields and/or **VIew**s. A **SPace** cannot, however, be used with the **OR** or **XOR** operators.

Field-Specific Definitions For Space

The **SPace** character can be defined within a specific field, field type or view. Whenever the same field name, type or view name is used within a **Find** order, the definition is active for that name.

Starting from the normal **SPace** default, suppose the following orders are given:

```
de vi receiver_name=rname ↵
Search  Records   Command
…
25      0         DEfine VIew receiver_name=rname
                  "VIEW RECEIVER_NAME redefined"
```

```
de sp(receiver_name)=and.s ↵
"Space within specified fields is now treated as
AND.S"
```

Followed by the search orders:

```
f receiver_name=mats lindquist ↵
Search  Records   Command
…
26      15        Find receiver_name=mats lindquist
```

```
f receiver_name=mats and.s lindquist ↵
Search  Records   Command
…
27      15        Find receiver_name=mats AND.S
lindquist
```

```
f rname=mats lindquist ↵
Search  Records   Command
…
28      1         Find rname=mats lindquist
```

```
f mats lindquist ↵
Search  Records   Command
…
29      1         Find mats lindquist)
```

```
f mats and Lindquist ↵
Search  Records   Command
…
30      50        Find mats AND lindquist
```

The first two orders are equivalent, and find all occurrences of the word 'Lindquist' in subfields of the field **rname** that also contain the first name 'Mats'.

The third order looks only for 'Lindquist' in subfields of the field **rname** where it is immediately preceded by 'Mats'.

The fourth order searches all fields for occurrences of 'Lindquist' where it is immediately preceded by 'Mats'. There is one occurrence in a **TExt** field.

The fifth order finds every occurrence of 'Mats' wherever it appears with 'Lindquist'.

Only those fields named in the definition of the **SPace** character will be affected.

Field names, types and views can be mixed in **DEfine SPace** orders as follows:

```
de sp(te,receiver_name)=and.f ↵
"Space within specified fields is now treated as
AND.F"
```

Following this order, these two **Find** requests would be equivalent:

```
f te=$rip system or receiver_name=mats lindquist ↵
Search  Records   Command
…
31      65        Find TExt=$rip system OR
                      receiver_name=mats lindquist
f te=$rip and.f system or receiver_name=mats
    and.f lindquist ↵
Search  Records   Command
…
32      65        Find TExt=$rip AND.F system OR
                      receiver_name=mats AND.F
lindquist
```

Comma and **SPace** may thus be given similar functions in **Find** orders; however, the comma cannot be restricted to selected fields, field types or views, and **SPace** cannot represent the **OR** operator.

Chapter 6:
Thesaurus Searching

In TRIP, a thesaurus is a database with a special structure, which permits it to be opened and used in conjunction with other databases in order to perform standard thesaurus operations. It contains words and phrases ordered according to their meanings, each of which is related to other terms in a treelike arrangement.

A thesaurus is a powerful tool which can enhance the quality of free text searching. It permits a user to extend TRIP's search functionality to include additional or alternative search terms, together with the terms used in a normal search strategy. These additional terms may be synonyms to a given term provided, or they may be terms which fall in a hierarchical tree structure that has been designed for a given application. TRIP permits the use of broader and narrower term relationships, as well as terms which are on the same level (called r*elated terms*).

In general, a thesaurus will have been developed by your system manager as part of an application. This chapter therefore only reviews the use of thesaurus commands and operators.

In order to effectively use the TRIP Thesaurus facility, it is important to understand the structure used by TRIP and how CCL search routines apply thesauri to a database.

Thesaurus Structure

Each thesaurus record contains the following elements and descriptions:

CT	(Controlled Term)	the main term of the record
BT	(Broader Term)	this term's nearest more general term(s)
NT	(Narrower Terms)	its nearest more specific term(s)
RT	(Related Terms)	relations other than the **NT** or **BT**
UF	(Used For)	synonyms and near-synonyms of the main term (**CT**)
SN	(Scope Note)	a description of the main term
NR	(Term Number)	a hierarchical term number (optional)

Figure 6–1 Thesaurus elements

Elements and Their Use

The Controlled Term

There is one main field in each record, the **CT** term (Controlled Term), which TRIP assumes is to be used when performing a thesaurus search. The Controlled Term is mandatory, and there can be only one per record. Its closest relations, the terms that occur in the **BT**, **NT**, and **RT** fields, will be the controlled terms of their own records in the thesaurus. The only indexed terms in a thesaurus are **CT**s and Synonyms. **BT**s, **NT**s and **RT**s are not indexed, since by definition they are references to other Controlled Terms. Any additional fields which may have been defined may or may not be indexed, just as in a standard database.

The Broader Term

Broader Terms are those which are one level up ('parent' terms) in the thesaurus tree. Multiple **BT**s are allowed for any **CT**, thereby permitting the thesaurus to branch out upward as well as downward. If a **CT** is actually the 'top term' or uppermost level in the tree, its **BT** will be left empty, and wherever a **BT** is used, it must also be the **CT** in another record. This is what forms the pathway up through the tree when searching.

For example, in the 'Animal' ladder below,

BT:
CT: Animals

BT: Animals .....
CT: Barnyard Animals

BT: Barnyard Animals .....
CT: Pig

the Controlled Term 'Pig' belongs to a larger group (its Broader Term) called 'Barnyard Animals'. It, in turn, is the Controlled Term of another record, and has 'Animals' as *its* Broader Term. 'Animals' itself forms the Controlled Term of yet another record, but because the Broader Term field of this record is empty, we know that 'Animals' is the top term of this tree.

Seen another way,

Animals
    Barnyard Animals
        Pig

'Pig' is a subset of 'Barnyard Animals', which is a subset of 'Animals', which is the top of the tree.

The Narrower Term

These are the terms one level below each Controlled Term, the 'child' terms. There can be many of these for every **CT**, and if the **CT** is at the bottom of the tree (the last term in its pathway), the Narrower Term will be empty. As with Broader Terms, for every Narrower Term used there must be a corresponding Controlled Term.

Using the previous example, 'Barnyard Animals' is a Narrower (lower) Term for 'Animals', and 'Pig' is a Narrower Term for 'Barnyard Animals'. 'Pig', however, having no **NT**s beneath it, is the lowest or most finely-divided term in the tree.

The Related Term

These are used to link Controlled Terms that are normally at the same level in the hierarchy ('sibling' terms), although it is possible to use related term links which have no clear hierarchical relationship to one another. There can be many Related Terms for each Controlled Term, but they are not required.

Adapting the 'Animal' example,

Animals

Barnyard Animals
- Pig
- Cow
- Horse
- Goat
- Sheep

Domestic Animals
- Dog
- Cat
- Rabbit
- Hamster
- Guinea Pig

Wild Animals
- Deer
- Fox
- Raccoon
- Opossum
- Rat

'Cow', 'Horse', 'Goat' and 'Sheep' may all be defined as Related Terms of 'Pig'. 'Barnyard Animals', 'Domestic Animals' and 'Wild Animals' may also be related terms of one another.

Defining and Using the Thesaurus

Since thesauri are standard TRIP databases, they can use all of the standard database functions discussed up to this point—they can be opened, searched, combined or joined with other databases, or used as controlled term lists for data entry in other databases.

A thesaurus only takes on a special character when the **DEfine THESaurus** command is applied to it. The **DE THES** command is a special case of the **DE MAP** (**DEfine MAP**) command; that is, thesaurus functions are basically a specialized set of indirect search commands (see Chapter Seven, 'Indirect Searching' for more information).

It is important to understand **DEfine THESaurus**, as it has a direct impact on search results.

When you **DEfine** a thesaurus in TRIP, you determine five things:

- the thesaurus that will be used (defined as active)
- the thesaurus element(s) to be searched (defaults: **CT** and **UF**)
- the thesaurus source element(s) from which to extract search terms (default: **CT**)
- the database destination field(s) to be searched for the extracted terms (default: all **PHrase** fields)
- whether or not exact matching of the entire phrase is required, i.e. if the subfields in the database(s) must contain nothing more than the phrase sought for (default: no).

Enter CCL Search mode and open database **Alice** to work the examples in this chapter:

```
bas alice ↵
```

TRIP gives the usual message:

```
Search  Records   Command
1       475       BASe alice
```

Hereafter, all examples showing a result in Search History will have that message printed immediately after the command which generated it.

You may recall having seen a number of databases with unfamiliar names while using the **BASe** command, one of which may have been **Thesali** (**THES**aurus for database **ALI**ce). If you give the **DEfine THESaurus** (short form: **DE THES**) order:

```
de thes=thesali ↵
Search  Records   Command
1       475       BASe alice
2       0         Define
                  THES=PHrase:thesali.CT:PHrase
```

TRIP notifies you that "THESALI is now the default thesaurus.", and that the three search defaults of the **DEfine** order are in effect. This TRIP shorthand can be interpreted as follows:

DEfine THES=PHrase:thesali.CT:PHrase

This portion of the DEfine statement indicates that only the indexed PHrase elements in the thesaurus will be searched (the default: CT and UF).

Additional possibilities include DEfine THES=CT…, where only the CT field is searched (no synonyms are used), and DEfine THES=PHrase+TExt…, in which all indexed PHrase and TExt fields are searched (CT, UF, SN and any other indexed fields present).

DEfine THES=PHrase:thesali.CT:PHrase

This portion of the order dictates which thesaurus is to be used.

DEfine THES=PHrase:thesali.CT:PHrase

This option determines which elements will be used as search terms, either against the target database (CT, BT, NT and RT commands) or in additional paths in the thesaurus (UP, DOWN and Display).

The default (shown above) is to extract terms from the CT element only. Another option is DEfine…PHrase, which allows direct term extraction from both Controlled Terms (CT), Synonyms (UF) and any other indexed PHrase fields.

DEfine THES=PHrase:thesali.CT:PHrase

This parameter controls which fields will be searched in the database which has been opened for searching by the terms found in the thesaurus, the default (shown above) being PHrase.

Other alternatives are DEfine…PHrase + TExt (or TExt + PHrase) which allows searching in both PHrase and TExt fields, DEfine…TExt, which limits searching to fields of type TExt, and DEfine…*field*, where *field* can be any single field or view name.

With the above **DEfine** defaults in place, if you were to give the order:

```
f ct(haigha) ↵
Search  Records   Command
…
3       15        Find CT(haigha)
```

TRIP will locate all the thesaurus records in **Thesali** that contain 'Haigha' in the **PHrase** (**CT** or **UF**) elements, pick the Controlled Terms (**CT**s) of these records, and look for them in all the **PHrase** fields of **Alice**.

You may wish to change these defaults.

For example, you might like to:

1. Direct the searches in the thesaurus to the **CT** element only.
2. Pick terms from the **CT** and the **UF** elements (**PHrase** elements).
3. Search for the terms in the **PHrase** and **TExt** fields of **Alice**.

To do this, give this **DEfine** order (using the short forms of the modifiers):

```
de thes=ct:thesali.ph:te+ph ↵
Search   Records   Command
…
4        0        Define
                  THESaurus=CT:thesali.PHrase:TExt+PHrase
```

After which the search request:

```
f ct(haigha) ↵
Search  Records   Command
…
5       16        Find CT(haigha)
```

will locate thesaurus records that contain 'Haigha' in their **CT** elements, pick the **CT** and the **UF** terms of those records ('Haigha' and 'King's Messenger'), and look for those terms in the **PHrase** and the **TExt** fields of **Alice**.

If you want exact matching of your search term (see item number five in the previous section, 'Defining and Using the Thesaurus'), enclose the field name or type(s) following the second colon in single quotation marks, as in:

```
de thes=ct:thesali.ph:'speaker' ↵
Search  Records  Command
…
6       0        Define
              THESaurus=CT:thesali.PHrase: 'speaker'
```

To view the thesaurus definitions, use the **DEfine** command

```
de? ↵
```

and scroll through the display to find the statement beginning with 'THES = '.

To restore the default settings, use

```
de thes=thesaurus_name ↵
```

Do this now with:

```
de thes=thesali ↵
Search  Records  Command
…
7       0        DEfine
                 THESaurus=PHrase:thesali.CT:PHrase
```

CCL Orders and the Thesaurus

CT and the other thesaurus operators are used in **Find** and **Display** orders to indicate that the current thesaurus is to be used when the order is carried out. The **Display** order will then show records from the thesaurus, and the **Find** order will look up the term in the thesaurus and search for the specified thesaurus terms in the open database(s).

Thesaurus Display

Building on the inherent structure within a TRIP thesaurus, the CCL module within MMC supports display and navigation within a tree (or set of trees) from that thesaurus. Additional information about each term within the tree is represented within a tooltip shown when you hover the mouse cursor above a given term. For example:

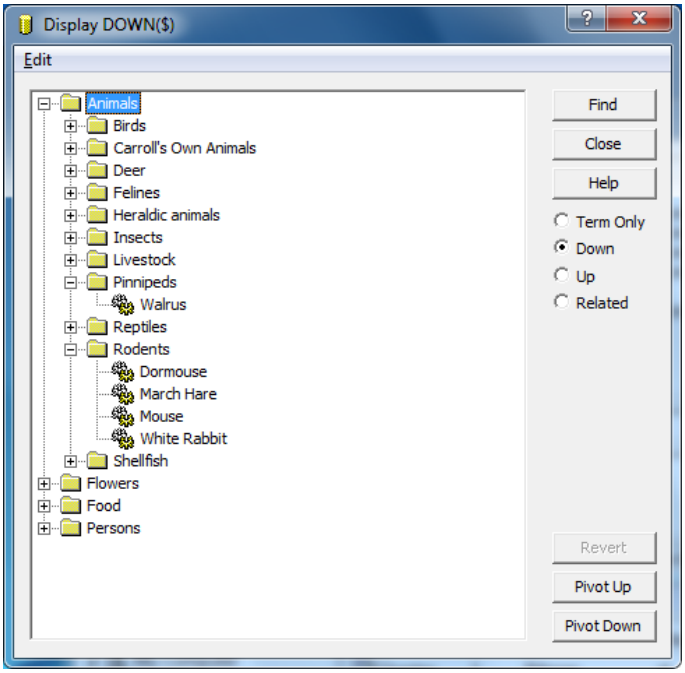

Figure 6–2 Display down($)

This set of trees is the result of a simple "Display DOWN($)" command using the THESALI demonstration thesaurus. As you can see, there are four distinct trees within this thesaurus, representing the genera "Animals", "Flowers", "Food" and "Persons". The tooltip that appears with a term reflects the UF, RT and SN fields from the term's record within the thesaurus.

Selecting a term and clicking the "Find" button will perform a search in the currently open database, leveraging the direction indication as shown in the set of radio buttons ("Down", "Up", etc.).

In addition, selecting a term and using the pivoting commands (the buttons at the bottom of the dialog on the right) will display the set of trees produced by navigating the thesaurus in the direction implied. Using the "Revert" command will re-display the original set of trees as produced by the first command issued by the user.

Warning!

> *Using the most generic commands within very large thesauri can result in memory usage that exceeds the capacity of the TRIPserver's process allocation. In such situations, the server process can terminate abnormally, resulting in an error display.*

Display searches the thesaurus and provides lists of terms based on the format used and the options included in the **DEfine THESaurus** command. **Find**, while it operates on the same principles as **Display**, also uses the list of terms collected as the search terms in the target (opened) database. We will make use of both throughout the remainder of this chapter.

For example, if the default **DEfine THESaurus** command is still in effect from the previous example (both **CT** and **UF** elements are searched, but only **CT** is extracted), the following six **Display** statements are true:

1) DISPLAY CT(*term*)
(where *term* can be any standard TRIP search value, including truncation) will find the *term* in all **CT** and **UF** elements and list it with some or all of this information (depending on how the thesaurus was built):

   - the Controlled Term
   - Synonyms (labelled ‘Syn:’)
   - Related Terms (labelled ‘RT:’)
   - a definition (labelled ‘Com:’, for ‘Comment’)
   - Try this Display CT request:

```
d ct(dinah) ↵
```

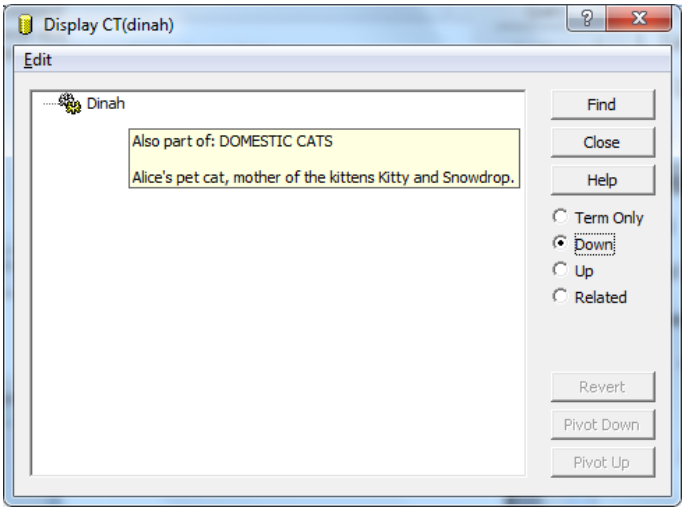

Figure 6–3 Display CT

TRIP informs you that it has found one main term (the Controlled Term; ‘Dinah’). Hovering the cursor over the tem displays a tooltip informing you that ‘Dinah’ is part

of the broader term, ‘Domestic Cats’ and that ‘Dinah’ is also Alice’s pet cat, the mother of the kittens ‘Kitty’ and ‘Snowdrop’ (the Definition or Comment).

2) DISPLAY BT(term)
will find the term in all CT and UF elements, extract all BT terms in those records, find CT terms which match them and display some or all of this information:

- the Controlled Term
- Synonyms
- the Broader Term (as a Controlled Term)
- Related Terms
- a definition

Levels within the tree are indicated by indentation of the Controlled Terms as they are displayed.

An example of a **Display BT** example might be:

```
d bt(haigha) ↵
```

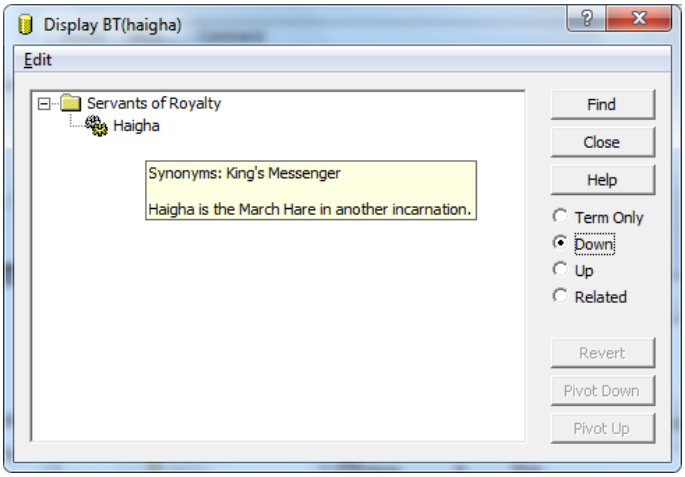

Figure 6–4 Display BT

There is one Controlled Term with a Broader Term of ‘Servants of Royalty’, while ‘Haigha’ is analogous to ‘King’s Messenger’ (Synonym) and the Definition or Comment states that ‘Haigha’ is actually the March Hare in another incarnation.

3) DISPLAY RT(term)
will find the term in all CT and UF elements, extract the RT terms, search for them in CT fields which match them and display any or all of the following:

- the Controlled Term
- Synonyms
- Related Terms
- a definition
- Try this exercise:

```
d rt(dodo) ↵
```

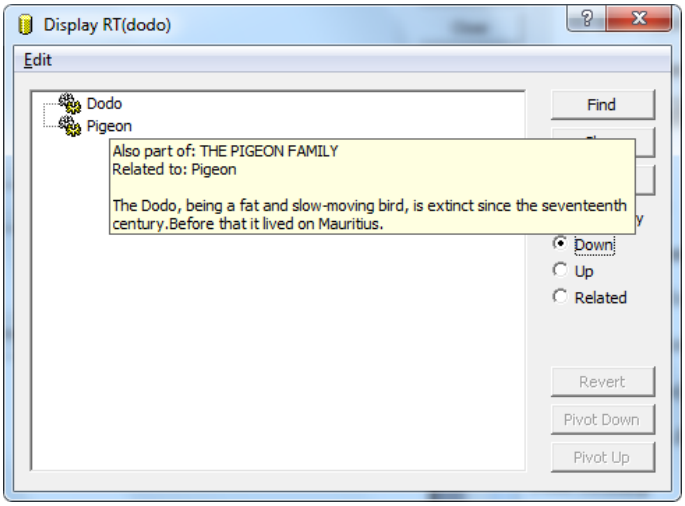

Figure 6–5 Display RT (Dodo tooltip)

There is, again, one main term ('Dodo') with 'Pigeon' as the Related Term. Dodo has a reference to the broader term 'The Pigeon Family', and a definition. There is also a cross-reference to the main term 'Pigeon', as it mentions 'Dodo' as a Related Term.

4) DISPLAY UP(term)
will find the term in the same manner as Display BT, except that all levels above the term specified are displayed, whereas Display BT only searches up one level. This means that the term is searched, its BT element is extracted and searched, then that term's BT element is extracted and searched and so on, until the top of the tree is reached. The display format is the same as for BT.

For example,

```
d up(footmen) ↵
```

and expanding the tree by clicking on any '+' symbols

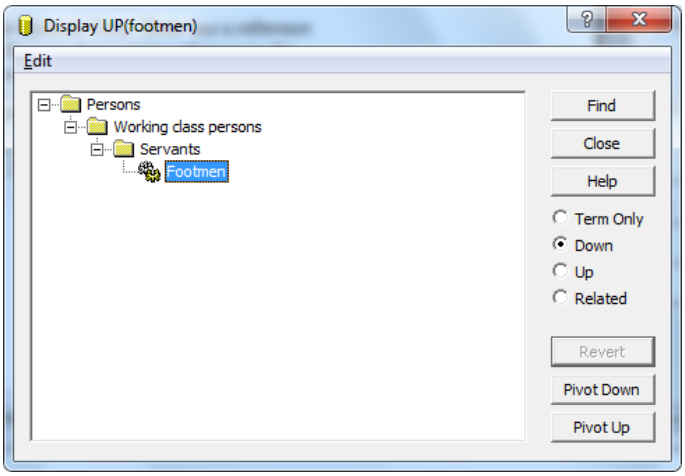

Figure 6–6 Display UP

informs us that 'Footmen' is a member of a larger group of terms called 'Servants', which is in turn part of a set known as 'Working class persons', which is an element of a main collection named 'Persons'.

5) DISPLAY NT(term)

and:

6) DISPLAY DOWN(term)
these work the same way as Display BT and Display UP, except that they apply the NT (Narrower Term) elements, thereby working downward from the term rather than upward. The display is also essentially the same.

Try this:

```
d down(footmen) ↵
```

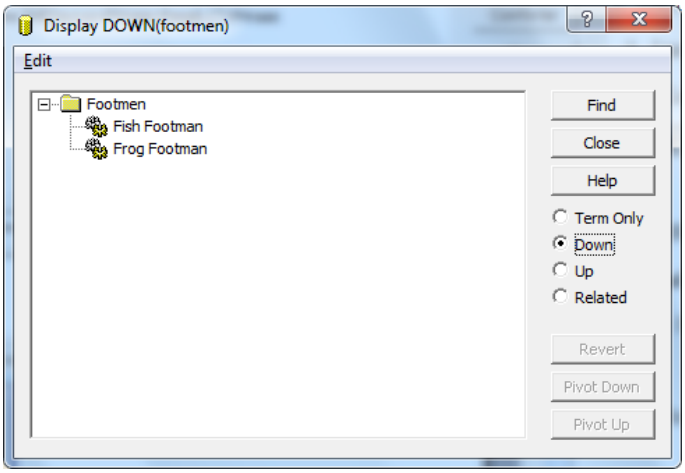

Figure 6–7 Display DOWN

We discover that the term ‘Footmen’ is in itself a larger grouping and that it contains two narrower terms, ‘Frog Footman’ and ‘Fish Footman’, each of which has the other defined as a Related Term (In this example, selecting the tooltips will just confirm the related terms).

Furthermore, it is possible to display any terms that may exist both above or below any term selected in the window by respectively clicking on the ‘Pivot Up’ and ‘Pivot Down’ buttons.

For example, selecting ‘Footmen’ and clicking on ‘Pivot Up’ will display all terms from ‘Footmen’ upward. Clicking on ‘Revert’ will return the display to it’s original state.

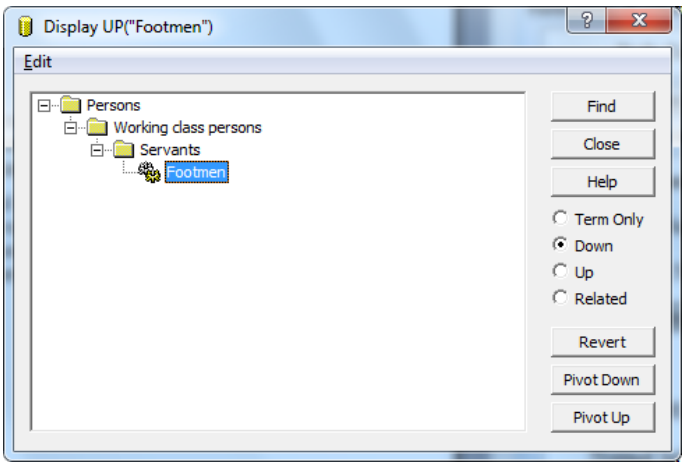

Figure 6–8 ‘Pivot Up’ from ‘Footmen’

Judicious use of these **Display** commands will help you to uncover and map the framework of, and relationships between, terms for any thesaurus.

Searching within a Thesaurus

There are two ways to **Find** terms within a thesaurus:

1) Entering **Find** commands directly into a CCL Query Session window.
2) Having TRIPmanager formulate the CCL **Find** commands for you by:
   a) performing a thesaurus **Display** command as described in the previous section
   b) selecting a term from within the term dsplay window
   c) choosing one of the four radio buttons, 'Term Only', 'Down', 'Up', or 'Related'; representing, respectively **Find CT(), Find UP(), Find DOWN()** and **Find RT()**
   d) clicking on the 'Find' button which will perform the relevant CCL **Find** command and add it to the history in your already open CCL Query Session window, from where you can issue (e.g.) a CCL **Show** command to look at the records hit.

Note:

In this guide, we will use CCL commands directly entered into a CCL Query Session, as this not only allows for greater brevity in the tutorial, it also allows you to gain greater familiarity with, and understanding of, the thesaurus operators.

With **Find**, TRIP searches for the *term* supplied using the same rules as discussed above. The default for the list of terms collected is Controlled Terms only, although other combinations of terms may be specified using other parameters in the **DEfine THESaurus** command. The search default is **PHrase** fields only in the target database. The result of each **Find** reflects the number of records which contain any of the Controlled Terms, and all **CT**s found are highlighted.

The thesaurus operators **CT**, **BT**, **UP**, **NT**, **DOWN** and **RT** may be used in **Find** commands as well as with **Display**, as these examples show. See the upcoming section on 'Searching Exercises' for further illustrations of their use.

```
f ct(fawn) ↵
Search  Records   Command
…
8       4         Find CT(fawn)
```

```
f bt(parrot) ↵
Search  Records   Command
…
9       5         Find BT(parrot)
```

```
f down(alice's servants)↵
Search  Records   Command
…
10      4         Find DOWN(alices servants)
```

```
f rt(stigand, archbishop of canterbury) ↵
Search  Records   Command
```

```
…
11      2          Find RT(stigand, archbishop of
canterbury)
```

Structure of Thesaurus Thesali

A thesaurus may be defined as having one or more main *trees* or pathways, each of which is actually a separate or sub-thesaurus. **Thesali** has four individual stems from which all other pathways originate, **Food**, **Flowers**, **Animals** and **Persons** as shown below
(… indicates 'More Terms'):

Thesali				
Food	Flowers	Animals	Persons	
Mutton	Rose	Birds	Persons…	
Pudding	Tiger-Lily		Passerines	Royalty
	…		…	…

Figure 6–9 Tree structure of thesaurus Thesali

Each tree is diagrammed below with its immediate sub-groupings.

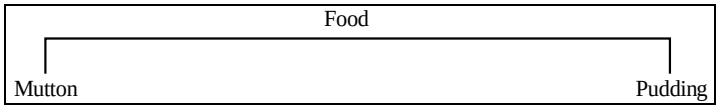

Figure 6–10 The 'Food' tree of thesaurus Thesali

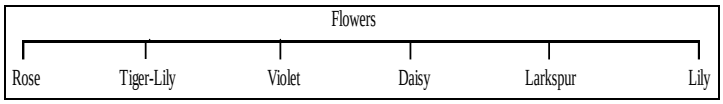

Figure 6–11 The 'Flowers' tree

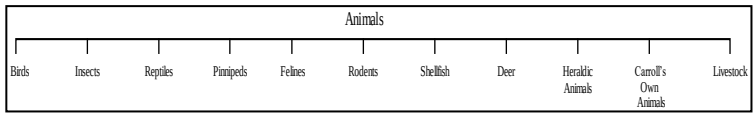

Figure 6–12 The 'Animals' tree

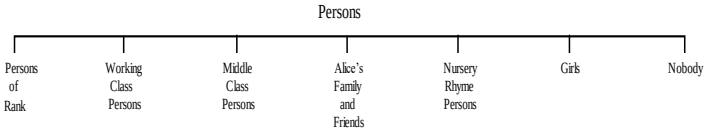

Figure 6–13 The 'Persons' tree

Exercises in Searching

Try the examples from **Alice** below:

Example 1:

```
f ct(#fly) ↵
Search  Records   Command
…
```

```
12      4          Find CT(#fly)
```

finds records having terms which end in 'fly' in the **CT** or **UF** elements, picks the **CT** for each of those records and searches for it in all **PHrase** fields in **Alice**.

Example 2:

```
f nt(alice's friends) ↵
Search  Records    Command
…
13      2          Find NT(alice's friends)
```

looks for all terms in the 'Alice's Friends' tree, and finds records containing the Narrower Terms (**NT**s) 'Ada' and 'Mabel'.

Example 3:

```
f up(alice's sister) ↵
Search  Records    Command
…
14      15         Find UP(alice's sister)
```

looks for all terms upwards from 'Alice's Sister' in that tree, and finds records with **PHrase** fields containing both 'Alice's Sister' and its Broader Term 'Alice's Family'.

Example 4:

```
f rt(edgar atheling) ↵
Search  Records    Command
…
15      2          Find RT(edgar atheling)
```

displays the records where 'Edgar Atheling' or its Related Term 'William the Conqueror' occur in **PHrase** fields.

The same truncating and masking characters can be used as in other **Find** and **Display** orders. The thesaurus records are numbered in the display, and a search order using these numbers will find the **CT** terms of the chosen thesaurus records in the **PHrase** fields of the database.

You may also restrict the searching in the thesaurus to exact matching of entire phrases by surrounding the search term in your order with single quotation marks.

Try the order,

```
f bee ↵
Search  Records   Command
…
16      6         Find bee
```

TRIP finds such expressions as ‘nowhere to *bee* seen’, ‘Busy *Bee*’, ‘a regular *bee*’, ‘*bee* good enough’, ‘*bee*-hive’ and ‘a single *bee*’. Now try:

```
f bt(bee) ↵
Search  Records   Command
…
17      29        Find BT(bee)
```

TRIP finds ‘Busy Bee’ as well as records with **PHrase** fields containing its Broader Term, ‘Insects’.

Try this order:

```
f bt('bee') ↵
A “No hits!” warning pop-up appears
```

This is because the Controlled Term entry in **Thesali** is ‘Busy Bee’—there is none for ‘Bee’!

You may also override the target specification of the **DEfine** order (**PHrase**, in these examples) temporarily by including a database field name in your search order. To search the field **txt** in **Alice** for the word ‘Caterpillar’ and its Broader Term ‘Insect’, use

```
f txt=bt(caterpillar) ↵
Search  Records   Command
…
18      12        Find txt=BT(caterpillar)
```

TRIP finds two hit terms, ‘Caterpillar’ and ‘Insects’.

Note:

*You may wonder why phrases such as ‘Looking-Glass Insects’ and ‘Insect’ were not found. This is because this search is directed toward the **txt** field, whereas ‘Looking-Glass Insects’ occurs in **chapter**, a **PHrase** field. Although there are many occurrences of the singular ‘insect’ in **txt**, since the **BT** of ‘Caterpillar’ is ‘Insects’, there is no match.*

Also, it is possible to write an order like

```
f bt(mouse) or bt('domestic cat') ↵
Search  Records   Command
…
19      17        Find BT(mouse) OR BT('domestic cat')
```

as

```
f bt(mouse or 'domestic cat') ↵
Search  Records   Command
…
20      17        Find BT(mouse OR 'domestic cat')
```

Chapter 7:
Indirect Searching

In its simplest form, indirect searching involves taking the vocabulary of one field from a set of records in one database and applying these as search terms to another database.

The ability to make indirect searches could be useful in a situation such as this:

A hypothetical medical database called **Cause&Effect** contains records which relate symptoms, disorders and cures, and another fictional database called **Case_Study** contains a large number of varied case studies.

A doctor has perused all of the case studies which mention **sleep disorders**, and now wishes to find all case studies which mention the **symptoms** of sleep disorders.

He or she can now use an indirect search to find all records in **Cause&Effect** which mention sleep disorders, and the contents of the **symptom** field of the hit records to search the **Case_Study** database.

The Indirect Search Process

Indirect searching takes place in three steps:

- find the desired set of records in the *source database* (the database from which the vocabulary terms are drawn)
- extract the contents of a single **PHrase** field (the *source field*) from all records found
- search for these terms in one or more fields of one or more *target databases* (those which are to be searched).

If you are not yet logged on to TRIP, do so now and choose CCL Search mode.

The DEfine MAP Order

Before giving an indirect search order, you must open the databases in which you want to search and specify the database name and field from which the search terms should be extracted. This is done using **DEfine MAP** (short form: **DE MAP**).

The simplest indirect search using this order is outlined in the series below:

```
bas databasename1 ↵

de map virtualfieldname =
        databasename2.fieldname ↵

f virtualfieldname(searchterm) ↵
```

- The first order (**BASe**) opens up the target database, *databasename1*.
- The second order (**DEfine MAP**) names a *virtual field*, which is a field which does not really exist for any database. The virtual field points to (or defines) the database (*databasename2*) and field within that database (*fieldname*) which will be the source of the vocabulary terms for the indirect search.
- The third order (**Find**) does several things:

- it searches the **PHrase** and **TExt** fields of *databasename2* for all records containing *searchterm*
- extracts the vocabulary contained in *fieldname for those records only*
- and searches *databasename1* for records containing any of those terms

Let's look at a practical example:

```
bas alice ↵

de map place=corr.scountry ↵

f place(mats) ↵
```

Following the process step by step:

- The first order opens the target database **Alice** (475 records).
- The second defines a virtual field called place which points to the field scountry in the source database Corr.
- The third order searches the PHrase and TExt fields of Corr (the default) for all records containing the name mats,

```
BASe corr          (99 records)
Find mats          (60 records)
```

pulls out the vocabulary terms stored in **scountry** for each of these records,

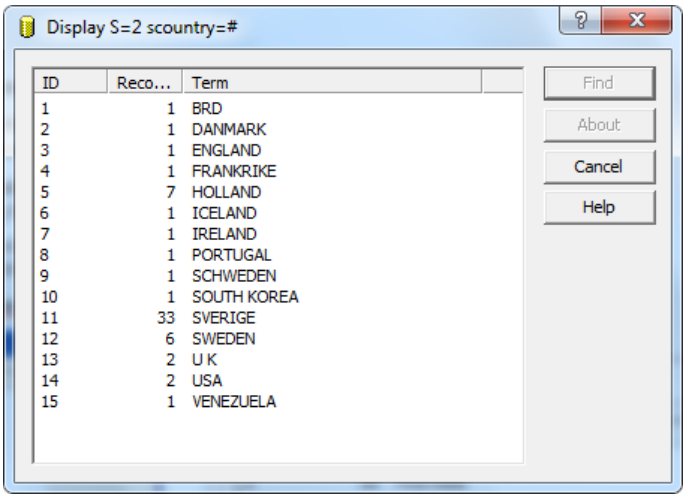

Figure 7–1 Display S=2 scountry=# example

and searches **Alice** for all records containing these terms,

```
Find brd OR danmark OR england OR frankrike OR
holland OR iceland OR ireland OR portugal OR
schweden OR u k OR usa OR venezuela          (2 records)
```

The third order can be considered as a function called **place( )**, with **mats** as the argument.

You will be able to view the vocabulary extracted from the **scountry** field of the **Corr** database by using the **Display** order:

```
d place(mats) ↵
```

which has this result (Figure 7 – 2):

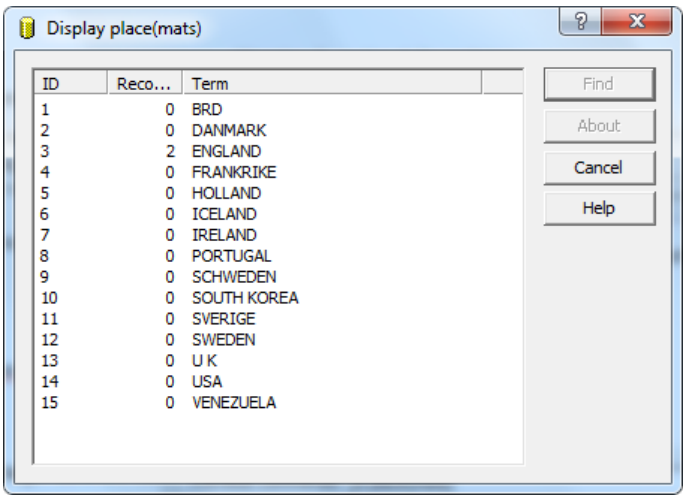

Figure 7– 2 Display place(mats) example

All terms except 'England' have zero occurrences, meaning that only 'England' produced hits in the target database.

The History box for this series contains:

```
Search  Records    Command
1       475        BASe alice
2       0          DEfine MAP

place=TExt+PHrase:corr.SCOUNTRY:TExt+
                        Phrase
3       2          Find place(mats)
```

In addition to the database and field names, the **DEfine MAP** order has several user-redefinable defaults similar to **DEfine THESaurus** (Chapter 6):

1. ***DEfine MAP*** place=***TExt+PHrase:***corr.SCOUNTRY:TExt+PHrase

This portion of the **DEfine** command states that TRIP will search only the **TExt** and **PHrase** fields of source database **Corr** for the search term requested (the name **mats**).

Additional possibilities include:

```
DEfine MAP virtualfieldname=TExt:...

DEfine MAP virtualfieldname=PHrase:...
```

where only the **TExt** or **PHrase** fields of the source database will be searched.

2. ***DEfine MAP*** `place=TExt+PHrase:corr.SCOUNTRY`***:TExt+PHrase***

This part of the order specifies that only the **TExt** and **PHrase** fields of target database **Alice** will be used for the final search.

Other options include:

```
DEfine MAP...:TExt

DEfine MAP...:PHrase
```

where either **TExt** or **PHrase** fields of the target database will be used for final searching.

Refer to the *CCL Reference Guide* for the full form of a **DEfine MAP** order, with its defaults and many variations.

To illustrate some other features of indirect searching, the examples which follow build on the first two steps of a **MAP**ped search as shown below:

```
bas carroll ↵
Search  Records   Command
…
4       24        BASe carroll

de map people=alice.person ↵
Search  Records   Command
…
5       0         DEfine MAP

people=TExt+PHrase:alice.PERSON:TExt+
                        PHrase
```

which open the target database **Carroll** (24 records) and create the virtual field **people** as a pointer to source database **Alice** and source field **person**.

The contents of History will be provided beneath each example in this chapter if appropriate.

Try the following orders:

Example 1:

```
d people(turtle) ↵
```

TRIP locates all the records in **Alice** which contain the term **turtle** in any **PHrase** or **TExt** field, extracts the contents of the field **person** for these records, sorts them alphabetically and displays them, giving the number of occurrences in **Carroll** (in any **PHrase** or **TExt** field):

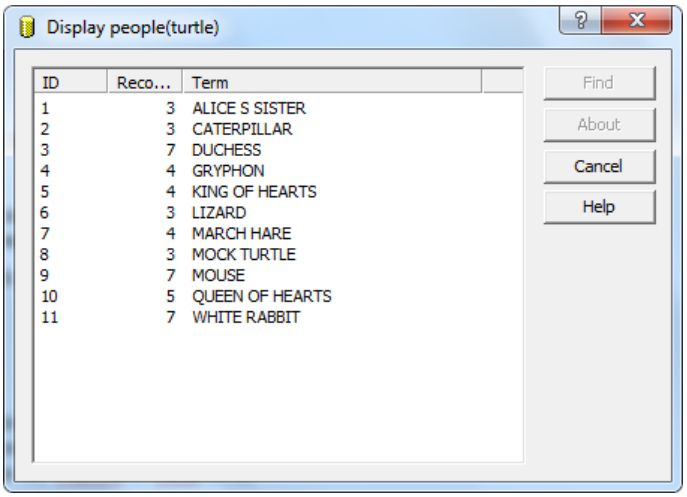

Figure 7– 3 Display people(turtle) example

Example 2:

```
f people(turtle) ↵
```

```
Search  Records   Command
…
6       14        Find people (turtle)
```

This order finds the terms displayed above in the **PHrase** or **TExt** fields of database **Carroll** and highlights them.

Example 3:

```
f verse=people(turtle) ↵
Search  Records   Command
…
7       2         Find verse=people (turtle)
```

looks for the terms displayed previously in **Carroll**'s **verse** field.

Example 4:

```
f speaker=people(turtle) ↵
Search  Records   Command
…
8       12        Find speaker=people (turtle)
```

finds the terms in the **speaker** field.

Example 5:

```
f speaker=people(turtle) and person=march hare ↵
Search  Records   Command
…
9       4         Find speaker=people (turtle) AND
person=march
hare
```

combines indirect and direct searching.

Example 6:

```
de map speakers=ph:alice.speaker ↵
Search  Records   Command
…
10      0         DEfine MAP

                  speakers=PHrase:alice.SPEAKER:TExt+PHrase
```

This new **DEfine MAP** order defines the virtual field **speakers** to map from the **speaker** field of **Alice** and specifies that searching in the source database **Carroll** only takes place in **PHrase** fields.

Example 7:

```
d speakers(knave) ↵
```

displays the vocabulary of the field **Alice.Speaker** for the records which contain the term **knave** in any **Phrase** field (Figure 7 – 4):

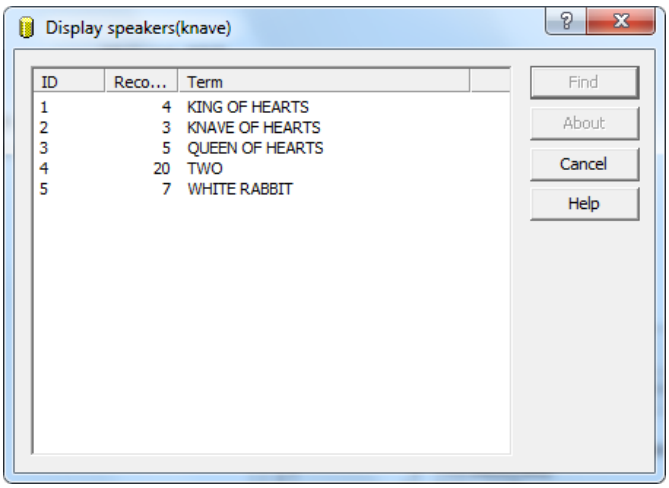

Figure 7– 4 Display speakers(knave) example

Example 8:

```
f speakers(knave) ↵
Search  Records   Command
…
11      21        Find speakers (knave)
```

finds those terms shown above in the database **Carroll** and highlights them.

Example 9:

```
de map speakers=ph:alice.speaker:text ↵
Search  Records   Command
…
12      0         DEfine MAP
 speakers=PHrase:alice.SPEAKER:TExt
```

This new **DEfine MAP** order defines the virtual field **speakers** to map from the **speaker** field of **Alice**. It also specifies that source database searching will occur only in **PHrase** fields, and that target database searching takes place in **TExt** fields.

Example 10:

```
f speakers(knave) ↵
Search  Records   Command
…
13      20        Find speakers (knave)
```

finds those terms shown above in **TExt** fields in the database **Carroll**.

Example 11:

```
f speakers(knave and heart# or queen# AND heart#) ↵
Search  Records   Command
…
14      21        Find speakers (knave AND heart) OR
(queen#
                  AND heart#)
f speakers(knave . hearts or dormouse) ↵
Search  Records   Command
```

```
…
15      20          Find speakers (knave . hearts OR
dormouse)
```

The *proximity operator* [**.**] in the latter example denotes 'separation of terms by zero or one term(s)'.

The argument of the indirect search function **speakers( )** does not have to be a single term, and may contain truncation characters, proximity and logical operators as shown above.

Exact Phrase Matching

There are two methods of ensuring that the extracted terms produce an exact match in the target database. An exact match means that the content of the subfield in the target phrase field contains nothing more than the **PHrase** or term extracted from the source **PHrase** field.

Exact mapping is indicated by single quotation marks, and could be included in the **DEfine MAP** statement as follows:

```
de map speakers=ph:alice.speaker:'phrase' ↵
Search  Records   Command
…
16      0         DEfine MAP
speakers=PHrase:alice.SPEAKER:'PHrase'
```

or included in a **Find** order:

```
f speakers('knave') ↵
Search  Records   Command
…
17      9         Find speakers ('knave')
```

The exact matching takes place in the target database in both orders.

Reference to an Earlier Search

The **S=n** qualifier is used to refer to the results of a previous search. Using the **Find** order in the source database which produced search result number three, the following order will reproduce a display list of the extracted terms:

```
d s=3.speakers ↵
```

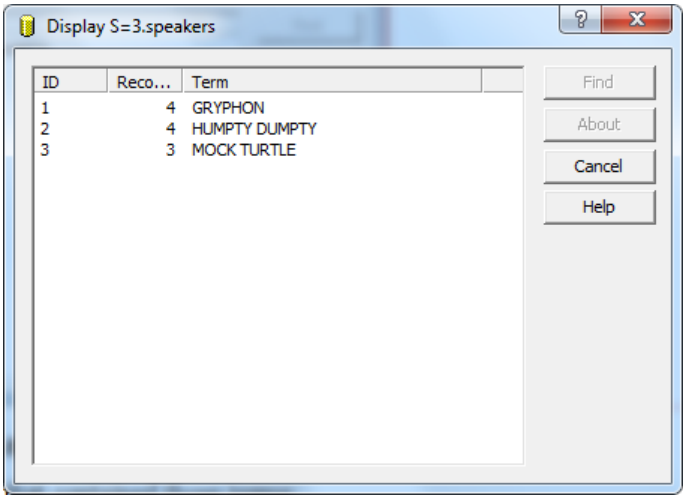

Figure 7– 5 d s=3.speakers example

and this one finds the records that contained those terms:

```
f s=3.speakers ↵
Search  Records   Command
…
18      4         Find S=3.speakers
```

In these cases, records were first located in the source database in search number three. The terms were then extracted from the **speaker** field in those records from the source database that are part of search result number three. The search and the display of the extracted terms in the phrase fields of the target database followed.

No argument is required.

Broadening a Search within a Database

As already mentioned, the target of an indirect search may be a cluster, i.e. a number of databases opened together which may contain the source database. Indeed, the target may be identical to the source, in which case each indirect search automatically and immediate broadens the search in the target database. This can be demonstrated using the **Alice** database and the following example.

Suppose the Jabberwocky is found murdered in his bed, and a list of suspects is required. The following orders would generate some leads:

```
bas alice ↵
Search  Records   Command
…
19      475       BASe alice
f jabberwock# ↵
Search  Records   Command
…
20      1         Find jabberwock#
```

This gives only one record, and the following order points the finger at ‘Humpty Dumpty’:

```
D s=20 person=# ↵
```

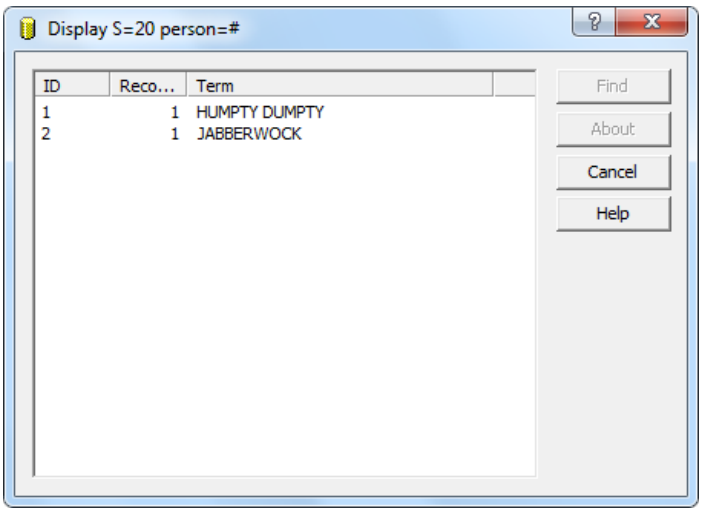

Figure 7– 6 d s=20 person=# example

By expanding the search using an indirect search order, a larger number of suspects is generated:

```
de map suspect=alice.person ↵
Search  Records   Command
…
21      0         DEfine MAP suspect=TExt +
PHrase:alice.PERSON:TExt+PHrase
f suspect(jabberwock#) ↵
Search  Records   Command
…
22      32        Find suspect(jabberwock#)
d s=22.suspect ↵
```

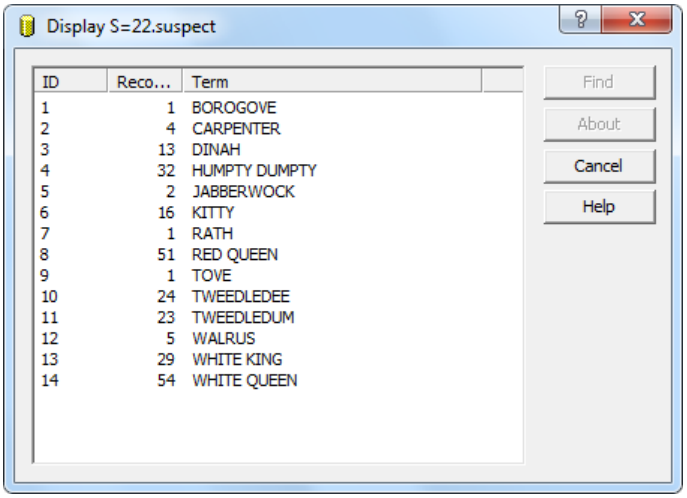

Figure 7– 7 d s=22.suspect example

Internal Transaction Sets

There is a simpler way to apply terms extracted from a search result set as search terms to the same database. In this kind of searching, the contents of one **PHrase** field (for a set of hit records) become the working or transaction set of terms for a further search. Although this works similarly to indirect searching from one database to another, as the source **PHrase** field can be specified directly in the **Find** or **Display** order, no **DEfine MAP** order is required.

Enter these CCL statements as preparation for the upcoming exercises:

```
bas corr ↵
Search  Records   Command
…
23      99        BASe corr
f #rip ↵
Search  Records   Command
…
24      86        Find #rip
f and system ↵
Search  Records   Command
…
25      60        Find S=24 AND system
f and data# ↵
Search  Records   Command
…
26      29        Find S=25 AND data#
```

Now consider these examples:

Example 1:

```
d rname=26.sname ↵
```

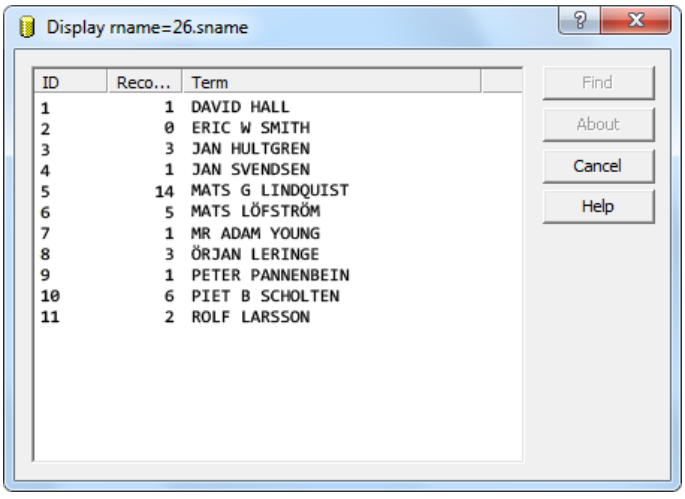

Figure 7– 8 d rname=26.sname example

This order extracts all of the terms in the **sname (sender)** field of the records located by the twenty-sixth search. The terms are then sorted alphabetically and presented in

a **Display** list, indicating for each term the number of records where it occurs in the **rname** (recipient) field.

Example 2:

```
f rname=0.sname ↵
Search  Records   Command
…
27      36        Find rname=26.sname
```

This order extracts the terms in sname in the records found by the most recent search, and searches for them in rname.

Example 3:

```
f (trip or tdbs) and te=0.rname ↵
Search  Records   Command
…
28      14        Find (trip OR tdbs) AND
                  TExt=27.rname
```

This order locates the records with the words **trip** or **tdbs**, as well as **TExt** occurrences of **rname** terms from the records found by the last search.

Example 4:

```
d 0.rname ↵
```

(Figure 7 – 9):

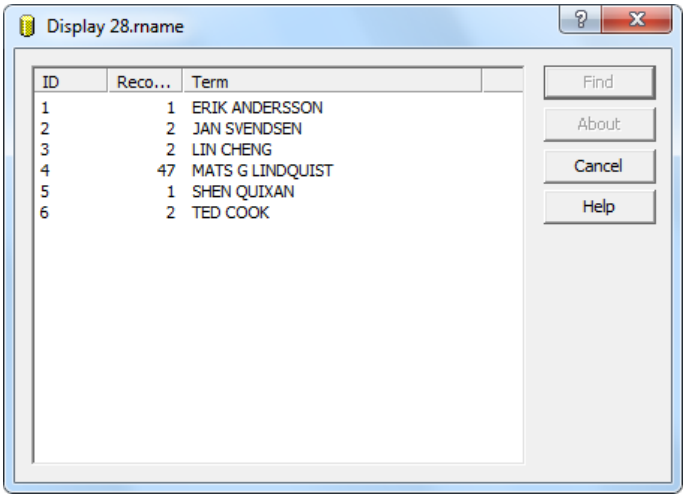

Figure 7– 9 d 0.rname example

This order displays a sorted list of all the different names in the field rname in the records of the latest search.

Example 5:

```
f 28.rname ↵
Search  Records   Command
…
29      49        Find 28.rname
```

This order locates all records that contain any one of the names in the **rname** field of the twenty-eighth search.

The last two **Find** and **Display** orders above are applied to **TExt** and **PHrase** fields as usual, unless this default has been changed by a **DEfine VIew** order.

External Transaction Sets

In the indirect search examples seen so far, the transaction set has been generated as the contents of a particular **PHrase** field, for a particular result set.

In TRIP, the transaction set can also be obtained from an ordinary text file, where each line of the text file constitutes a term. This search function was introduced to enable TRIP searching with terms acquired from searches in other systems, and can also be used to store search profiles applicable to several searches.

The form of a **Find** or **Display** order is follows the standard format for thesaurus and indirect searching orders. The modifier **FILe** signals the use of an external transaction set, and the name of the file is surrounded by parentheses.

Here is a sample **Find FILe** order using a file called 'Mylist.DAT':

```
Find FILe(mylist.dat) ↵
```

which contains a list of terms to be searched for. Indirect searching of this kind can be used in the same way as other search orders:

```
Find PHrase=FILe(mylist.dat) AND day>1986-10-31 ↵

Find S=0 OR new_name=FILe(mylist.dat) ↵

Display rname=FILe(mylist.dat) ↵
```

This is ordinary text searching, and the usual rules and defaults apply.

The specification of the filename may vary depending on the host operating system. However, if the file is not in the directory from which TRIP was started, the full file path name should be included. For example:

```
Find FILe(/users/myuser/mylist.dat) ↵

Find FILe(C:\mydir\mylist.dat) ↵
```

are examples of UNIX and Windows pathnames, respectively.

Note:

The above mentioned file paths point to files that exist on the TRIPsystem server *and not on the local system.*

Maximum Limits

Indirect **Find** and **Display** orders are governed by the same maximum limits as direct **Find** and **Display** orders. The number of terms extracted from the **PHrase** field of the source database is also governed by the **Display** limit.

The number of hit records in the source database from which these terms are drawn also has a limit. The initial value is 1000, but it may be changed using a **DEfine MAP MAXimum** order (short form: **DE MAP MAX**) such as:

```
de map max=2000 ↵
```

TRIP responds with the message "New limit for MAP set to 2000."

Chapter 8:
Head and Part Records

A record may exist as a two-level tree structure, composed of a *head* or *main record* and any number of *part* or *sub-records*. If head and part records have been included in the design of any particular database, each field is by definition either a head or a part field for that database.

The head record, which includes the *head fields*, is described by the contents of these fields and generally contains information which is relevant to all its part or sub-records. The part records (each of which holds one or more of the *part fields*) are described by the contents of those fields, and usually contain information which is applicable to that sub-record only. Main records with sub-records are illustrated in the diagram that follows.

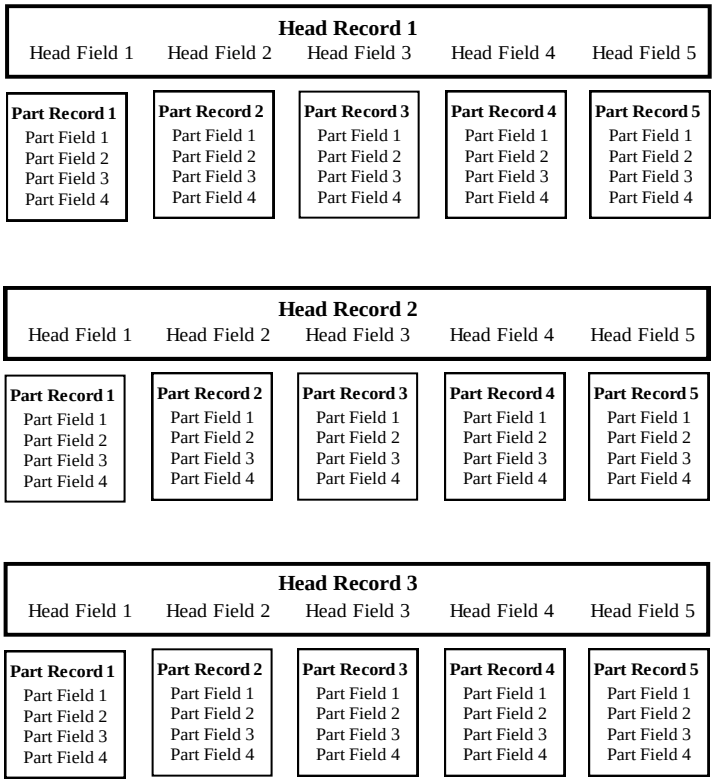

Figure 8–1 The relationship between head and part records

We will use the demonstration database **Carroll** as an example.

Log on to TRIP, enter **CCL Search** mode and request to see the structure of this database with

```
st carroll ↵
```

As you can see in the figure below, the content of this database is similar to that of database **Alice**, however the information is arranged a bit differently. Head records made up of head fields now contain all of the *chapter information*—number and heading, the list of persons acting in the text and a new field, the title of the book from which the text was taken. The part fields within the part records hold the *paragraph information*, and include the speakers in the text as well as all of the text fields.

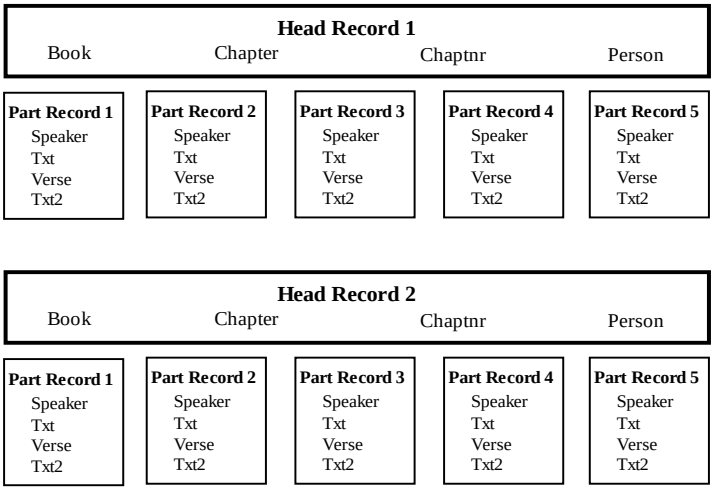

Figure 8–2 Carroll's head and part record structure

Each of the twenty-four main (chapter) records in **Carroll** now has from one to thirty-seven sub-(paragraph) records clustered beneath it.

Defining Terms

In order to describe some of the search techniques, we must first define some terminology regarding head and part records. Consider the record outlined in Figure 8–3, which has five part records.

The terms used for records with sub-records are:

- The head record or record head, which is made up of all of the head fields.

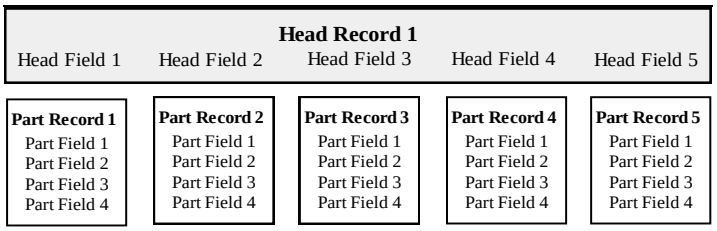

Figure 8–3 A head record

- The part record or record part, consisting of one set of part fields.

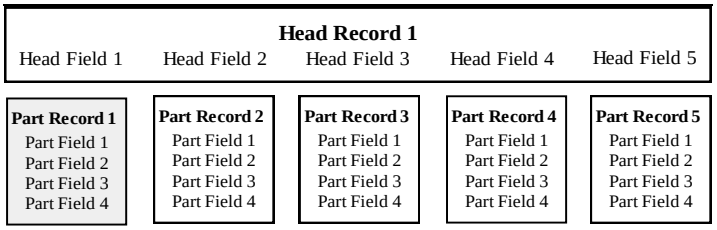

Figure 8–4 A part record

- Record entity 1, 2, 3 etc. represent the union (or sum) of head record 1 and part record 1, head record 1 and part record 2, head record 1 and part record 3, etc.

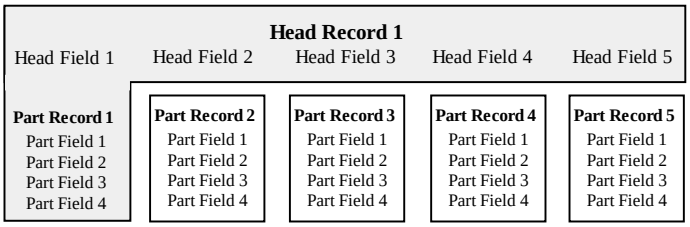

Figure 7– 10 A record entity

- A record is the union of the head record and all of its part records.

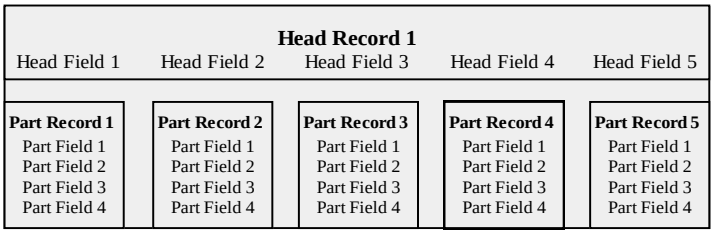

Figure 7– 11 A record

- Components are the unit records (individual head and part records) in the database.

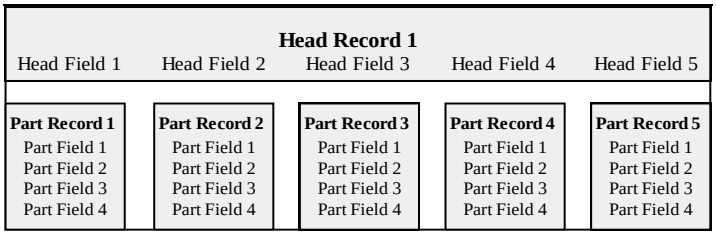

Figure 7– 12 Components of a record

Searching

Several operators not previously discussed make searching of databases with part records more discriminating. These are:

Operator	Signifying	
AND.R	AND	within a record (head + all parts)
NOT.R	NOT	within a record (head + all parts)
AND.E	AND	within an entity (head + one part)
NOT.E	NOT	within an entity (head + one part)
AND.C	AND	within a component (head or part)
NOT.C	NOT	within a component (head or part)

Figure 8–5 Part record operators

Examples

To work the following exercises, open database **Carroll** using:

```
bas carroll ↵
Search  Records   Command
1       24        BASe carroll
```

Next, give the order:

```
f white bread and.r brown bread ↵
1       24        BASe carroll
2       1         Find white bread AND.R brown
bread)
```

which finds all records where the terms ‘white bread’ and ‘brown bread’ are mentioned anywhere in the same record, as it would in a database without part records.

Now try:

```
f looking glass and.e insect ↵
Search  Records   Command
…
3       1         Find looking glass AND.E insect
```

This example finds all records in which the terms ‘looking glass’ and ‘insect’ are found in any head + part record pairing or entity.

DEfine this **VIew**:

```
de vi view_people=person,speaker ↵
Search  Records   Command
…
4       0         DEfine VIew view_people=person,
speaker
```

and type in:

```
f view_people=white knight and.e red knight ↵
Search  Records   Command
…
5       1         Find view_people=white knight
AND.E
                      red knight
```

This request finds all records containing ‘White Knight’ and ‘Red Knight’ in a head record/part record pair, either in the **person** field (head portion) or the **speaker** field (part portion).

Make this request:

```
f rabbit and.c kid gloves ↵
Search  Records   Command
…
6       2         Find rabbit AND.C kid gloves
```

which finds ‘rabbit’ and ‘kid gloves’ in the same head or part record.

The operators **AND** and **NOT** are by default equivalent to **AND.E** and **NOT.E**, but you may change this with one of several **DEfine** orders:

```
de and=and.r ↵

de and=and.c ↵

de and=and.e ↵
```

These statements change the default for both **AND** and **NOT** so that they will always work within the same domain, whether it be entities, records, or components.

Output and Sorting

Output

The default report for databases employing part records has been programmed to output *all* sub-records of a hit record in search results, not only those which include hit terms. This can make browsing the output from a search result unwieldy if each head record contains many parts and you are not using **Show FOcus**. To make browsing easier it is possible to predefine one or more reports, so that a head-part database will output only part records containing hit terms. If there is a database which uses head and part records at your installation, contact your Application Manager to see if this option is available.

Sorting

If your **Show** request includes a **SORt** on head fields only, sorting is performed as expected, on the fields and in the manner you request. If you wish to **SORt** on one or more part fields, there are two options:

1. The **SORt** modifier, which sorts records according to the contents of the field in the normal way and outputs records in the order required.

2. The **PART SORt** (short form: **PART SOR**), which sorts entities according to the content of the part records and outputs entities sorted on part records.

A 'general' **SORt** uses the usual sort function (in combination with the default report) and lists each head followed by all parts, regardless of whether or not the target field contains information.

A **PARt SORt** arranges each record by *entity*, and displays only those parts which include data.

In special cases the System Manager may have set up a 'Hit List' function, which specifies that only the parts containing hit terms will be output.

There may be a complex relationship between sort functions and reports—see your System Manager for further details.

To illustrate, type in this statement:

```
f speaker=(mouse or dormouse) not speaker=mad
hatter ↵
Search  Records   Command
…
7       4         Find speaker=(mouse OR dormouse)
NOT
                      speaker=mad hatter
```

TRIP finds four records (with a combined total of fifteen part records) that match the criteria given in the order above.

When the search result is output using the normal **SORt** modifier (Figure 8-61, overleaf, at left), all fifty part records having entries in the **speaker** field are included, whether or not they contain hit terms.

When the output is generated using **PARt SORt** (Figure 8-61, right), however, only the fifteen hit parts are shown. For emphasis, hits are shown in boldface and italics.

`Show SORt=speaker format=speaker`

Record–Part	Record–SPEAKER:
2–3	White Rabbit
2–5	Alice's Family
2–12	***Mouse***
2–13	***Mouse***
2–14	***Mouse***
3–1	Lory
3–2	***Mouse***
3–3	***Mouse***
3–4	Lory, ***Mouse***, Duck
3–5	***Mouse***
3–6	Dodo, Eaglet
3–7	Dodo
3–8	Dodo
3–9	Dodo, ***Mouse***
3–10	***Mouse***
3–12	***Mouse***
3–13	***Mouse***, Lory, Crab, Crab's daughter
3–14	Lory, Magpie, Canary
7–1	March Hare, Mad Hatter
7–2	March Hare
7–3	March Hare, Mad Hatter
7–4	March Hare, Mad Hatter, Dormouse
7–5	March Hare, Mad Hatter
7–6	March Hare, Mad Hatter
7–7	Mad Hatter, Dormouse
7–8	March Hare, Mad Hatter
7–9	March Hare, Mad Hatter
7–10	Mad Hatter
7–11	Mad Hatter, Dormouse
7–12	Mad Hatter
7–13	March Hare, Mad Hatter, Dormouse
7–14	***Dormouse***
7–15	March Hare, Mad Hatter
7–16	March Hare, Mad Hatter, Dormouse
7–17	Mad Hatter, Dormouse
7–18	Mad Hatter, Dormouse
7–19	Mad Hatter, Dormouse, March Hare
11–4	Gryphon, White Rabbit
11–6	King of Hearts, White Rabbit
11–7	King of Hearts, Mad Hatter, White Rabbit
11–8	King of Hearts, Mad Hatter, White Rabbit, Dormouse, March Hare
11–9	King of Hearts
11–10	***Dormouse***
11–11	***Dormouse***
11–12	King of Hearts, Mad Hatter

`Show PART SORt= speaker format=speaker`

Part	SPEAKER:
3–9	Dodo, ***Mouse***
7–14	***Dormouse***
11–10	***Dormouse***
11–11	***Dormouse***
11–18	King of Hearts, Cook, White Rabbit, ***Dormouse***, Queen of Hearts
3–4	Lory, ***Mouse***, Duck
3–13	***Mouse***, Lory, Crab, Crab's daughter
2–12	***Mouse***
2–13	***Mouse***
2–14	***Mouse***
3–2	***Mouse***
3–3	***Mouse***
3–5	***Mouse***
3–10	***Mouse***
3–12	***Mouse***

11–13	King of Hearts, Mad Hatter, March Hare
11–14	Mad Hatter, King of Hearts
11–16	King of Hearts, Mad Hatter, Queen of Hearts
11–17	King of Hearts, Cook, White Rabbit
11–18	King of Hearts, Cook, White Rabbit, ***Dormouse***, Queen of Hearts
11–19	King of Hearts, White Rabbit

Figure 8–6 PARt SORt output

You will notice that the contents of **speaker** apparently were not alphabetized in the example using normal **SORt** (top of chart, left). These terms were indeed sorted, but perhaps not in the manner one would expect.

When using normal **SORt** on part fields, TRIP first examines the first term in each target part field in a hit record. It then alphabetizes the hit record using *only that term*. Using Record 2 from the example above (at left), TRIP scans the 'Speaker' list in each part record:

Record 2

Part 3:	SPEAKER:	White Rabbit
Part 5:	SPEAKER:	Alice's family
Part 12:	SPEAKER:	***Mouse***
Part 13:	SPEAKER:	***Mouse***
Part 14:	SPEAKER:	***Mouse***

alphabetizes the part record list:

Record 2

Part 5:	SPEAKER:	Alice's family
Part 12:	SPEAKER:	***Mouse***
Part 13:	SPEAKER:	***Mouse***
Part 14:	SPEAKER:	***Mouse***
Part 3:	SPEAKER:	White Rabbit

and selects the contents of the first field in the alphabetized list as the 'part field representative' for that hit record, to be used in sorting the final list:

Record 2

Part 5:	SPEAKER:	Alice's family
Part 12:	SPEAKER:	***Mouse***
Part 13:	SPEAKER:	***Mouse***
Part 14:	SPEAKER:	***Mouse***
Part 3:	SPEAKER:	White Rabbit

If two or more part fields in a part record have identical contents (for example, Parts 12–14 above all contain 'Mouse'), then the parts are ordered by part number.

Here are the final part field representatives for each of the four hit records from the previous example:

Record 2, Part 5:	SPEAKER:	Alice's family
Record 3, Part 7:	***SPEAKER:***	***Dodo***
Record 7, Part 14:	***SPEAKER:***	***Dormouse***
Record 11, Part 10:	***SPEAKER:***	***Dormouse***

If two or more part records have identical part field representatives (for example, Parts 10 and 14 above are both 'Dormouse'), then the records are ordered by record number.

The main points to remember regarding head/part records are:

- TRIP finds only entire records; not head records or part records, but whole records.
- The **PART SORt** modifier sorts and shows the output by entity, which is neither output nor used for sorting if it is not included in the hits.

Part 3:

Data Entry

Chapter 9:
Data Entry

In TRIPclassic, data entry forms are used for adding records to, modifying records in and deleting records from a database, one record at a time. A data entry form is simply a 'window' into a database and consists of literals (fixed text) and a number of entry boxes, each of which is linked to a field in the database to which it writes.

Notes:

- *Actual data entry is currently unavailable in TRIPmanager. It is, however, possible to perform basic administration of Data Entry forms.*
- *You can only manage those forms to which you have access.*

Listing Available Data Entry Forms

To view the existing data entry forms for a given database, expand its sub-tree in the mmc window; then click on the 'Entry Forms' icon. A list of entry forms will then appear.

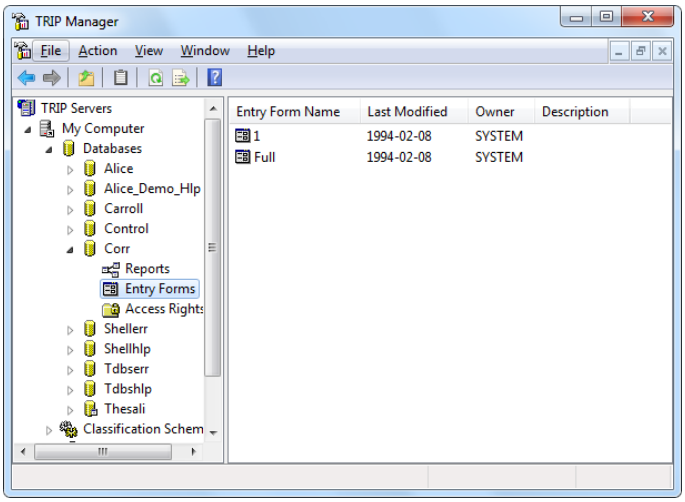

Figure 9–1 Entry forms for database CORR

Selecting an entry form and choosing 'Properties' from the action menu, will list the properties for that form.

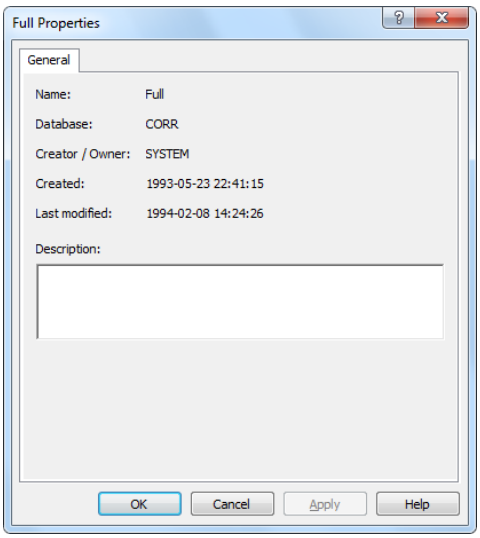

Figure 9–2 Properties for CORR entry form FULL

Creating and Modifying TRIPclassic Data Entry Forms

Unfortunately, TRIPmanager currently has no means of carrying out these operations. It is hoped to include this functionality in a later release. For now, consult the *TRIPclassic User Guide* for details on how to carry out these tasks.

Copying TRIPclassic Data Entry Forms

To copy a data entry form, click on 'Entry Forms' in the chosen database sub-tree, select the form to copy and select 'Copy' from the action menu. Next, select 'Paste from the action menu, with the cursor in the right hand window of the mmc.

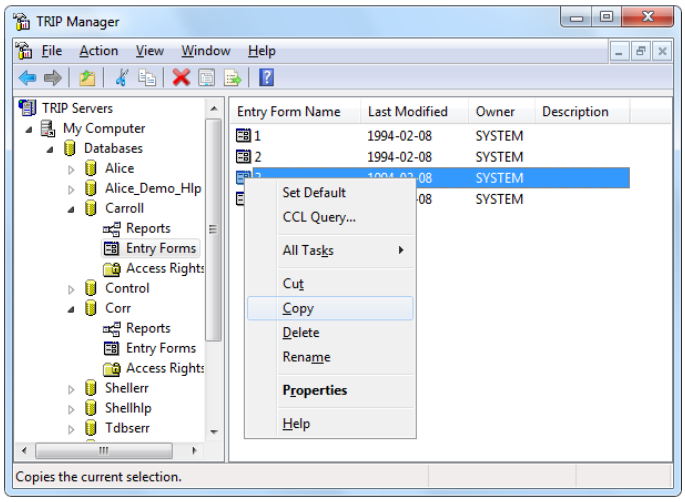

Figure 9–3 Copy a Data Entry form

A dialogue will appear, in which you may enter the name for the new Entry form copy and click on the 'OK' button to confirm the action.

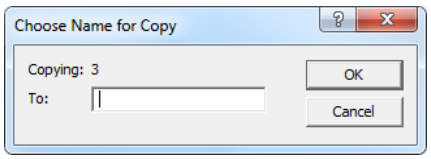

Figure 9–4 Name New Data Entry Copy

a new Entry Form creation confirmation will then appear.

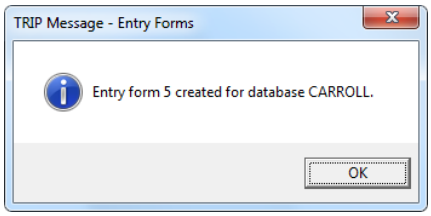

Figure 9–5 Data Entry Copy Confirmation

Clicking on the 'OK' button will clear the confirmation. The copy of the new form has now been created.

Deleting TRIPclassic Data Entry Forms

To delete a data entry form, click on 'Entry Forms' in the chosen database sub-tree, select the form to copy and select 'Delete' from the action menu.

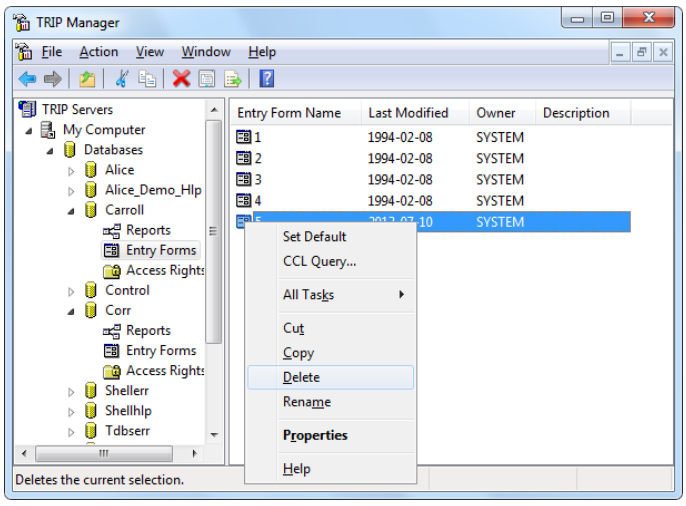

Figure 9–6 Delete a Data Entry form

A 'Yes/No' confirmation dialogue will appear. Click the 'Yes' button to confirm the action, or the 'No' button to cancel it.

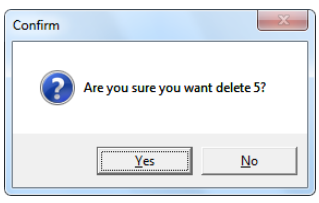

Figure 9–7 Delete Data Entry form confirmation

Clicking on 'Yes' will cause a deletion confirmation to appear.

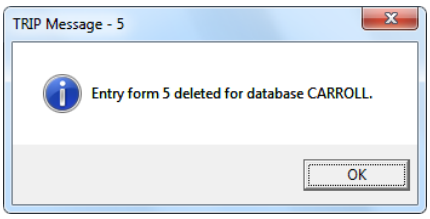

Figure 9–8 Data Entry form Deleted

Clicking on the ‘OK’ button will clear the confirmation. The selected data entry form has now been deleted.

Part 4:

User Administration and Procedures

Chapter 10:
User Administration

This chapter covers two elements of the user environment which are under user control, the **password** and **profile**.

Changing Your Password

Select the database connection for which you wish to change the user's password and chose 'Properties' from the action menu.

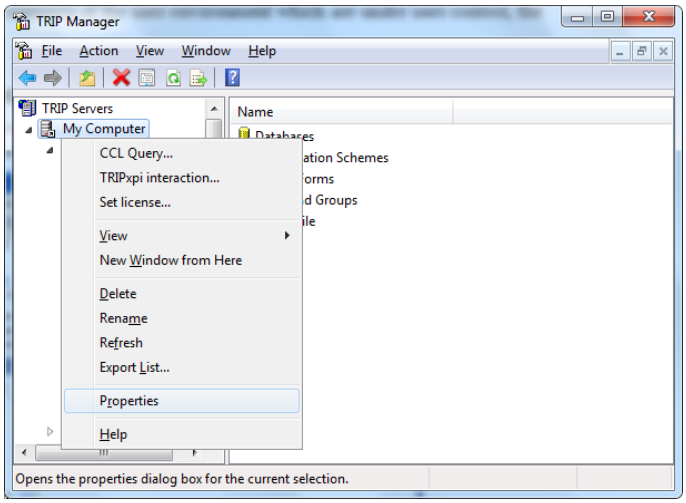

Figure 10–1 Selecting the database properties

The **Properties** window appears with your User Identification pre-entered in the field marked 'TRIP user name'.

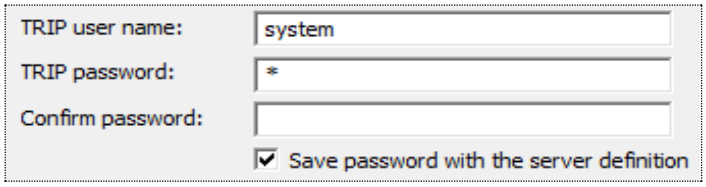

Figure 10–2 The Change Password dialogue

The password entry boxes will remain greyed out until you select the 'Save password with server definition' checkbox. Check this checkbox and enter your password into the 'TRIP password' entry box. Re-enter the password into the 'Confirm password' entry box and click on 'OK' to save the changes.

Note:

Passwords can be from one to thirty-two characters long, with no spaces.

You can also set up your User Profile so that you do not need to enter a password, if your TRIP username is the same as your local system username (Local System Validation), or if LDAP has been enabled on your network and TRIP installation.

Please see the section below on the '*User Profile: Login Type*' for a brief description of enabling Local System Validation for a user and '*Appendix B: TRIP User Account*

Validation Methods' of the *TRIPmanager Administration Guide* if you wish to have more detail on either facility.

User Profile

The User Profile window allows you to change some of the ways in which TRIP responds to your actions.

To see your User Profile, select the 'My Profile' tree item and select 'Properties' from the 'Action' menu..

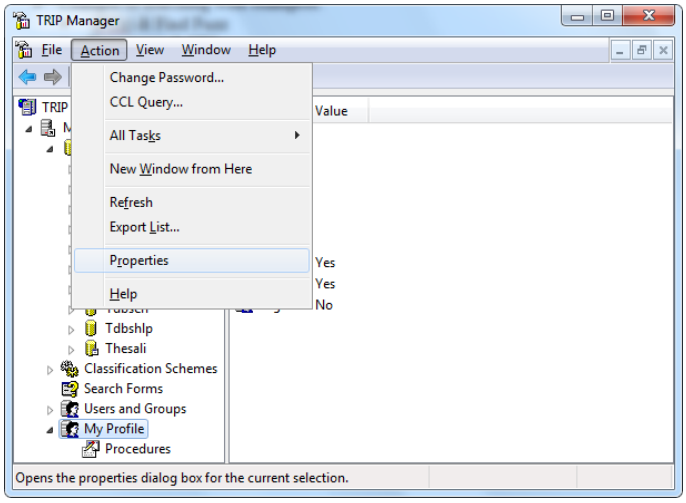

Figure 10–3 Selecting User Properties

The **User Properties** screen will appear:

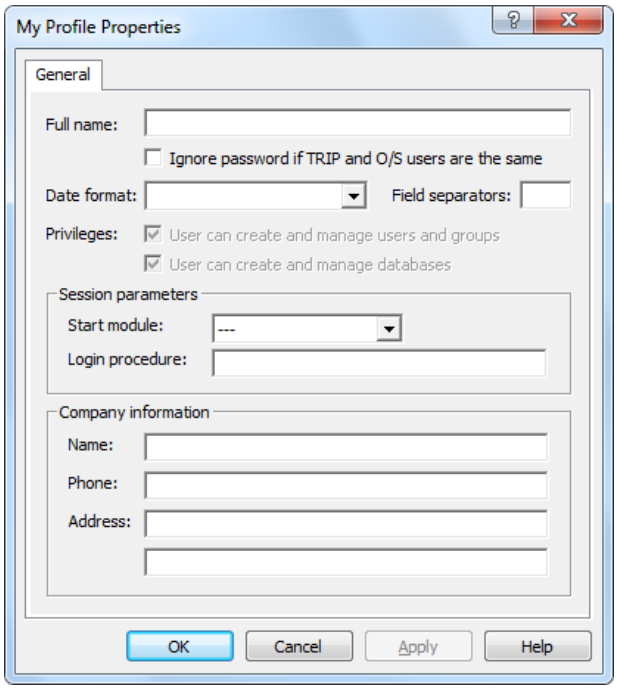

Figure 10–4 The User Properties window

The User Profile attributes are divided into sections:

Full Name

Login Type
Date Form
Start Procedure and Module
Company Information
User Privileges (Greyed out)

Each of these is now covered in turn.

Full Name

You may enter your full name into the 'Full Name:' entry box if you wish.

Note:

This field is not for entering your username: It is for descriptive purposes only.

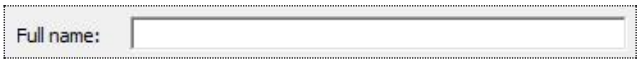

Figure 10–5 Full Name

Login Type

If your TRIP username is the same as your Host or system username, and the 'Ignore password if TRIP and O/S users are the same' checkbox is selected, you will not need to enter your password when you log on to TRIP with that username.

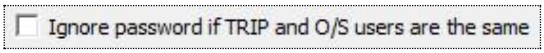

Figure 10–6 Login Type

Date Form

This group of fields allows you to change your date form, both the format in which the date is entered into fields on search and entry forms, and that in which it is displayed on output (unless the form is fixed by the report itself). Both format and separators (characters which divide the month, day and year) may be changed.

To change the date form, position the cursor to the immediate left of 'for May 1st, 1988'.

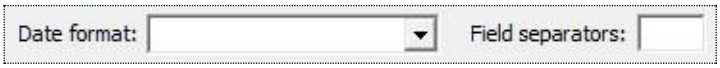

Figure 10–7 The Date Format boxes

To change the digit separator characters, chose the desired format from the 'Date format:' drop-down selection box. If you wish, you can enter your own field separator character into the 'Field separators:' entry box, using a hyphen [-], slash [/], full stop [.] or full colon [:].

Note:

*In addition to the date form in your profile, TRIP always recognizes the eight-character form **CCYYMMDD** (no separators between century, year, month and day), which gives **19880523** for **May 23, 1988**.*

Start Module and Login Procedure

Start Module

The **Start Module** options in this drop-down selection box are, ‘CCL Search’, ‘Form Search’ or ‘…’ (none).

Figure 10–8 Start Module

To start every TRIPclassic session at the command line of the CCL Search window, choose the ‘CCL Search’ option from the drop-down selection box. If you select the ‘Form Search’ option, TRIP begins the session by asking for a search form. Selecting ‘…’ does not select a Start Module.

Login Procedure

If the name of a procedure is entered in the **Login Procedure** entry box, that procedure will be executed when you start a TRIPclassic session and enter CCL Search mode for the first time. This can be any type of procedure to which you have access, whether private, group or public.

Figure 10–9 Login Procedure

For example, if you would like to run the same search (such as an SDI search) or define a particular entry form to be the default every time you begin a TRIP session, you can specify that here.

Company Information

These first four lines allow you to enter your employer’s name and address and your telephone number if desired.

User Privileges

These fields tell you what ‘manager rights’ you have in TRIP. However, the ability to create new databases, users, groups of users etc. is assigned by the system manager and cannot be changed by the user; hence the boxes are greyed out unless you are system manager.

Figure 10–10 User Privileges

A tick in the ‘User can create and manage users and groups’ checkbox indicates that you are able to create new users and user groups. A tick beside the ‘User can create and manage databases’ checkbox allows you to create, modify and grant user and user group access to your own databases.

Chapter 11:
User Procedures

Procedures are used to automate TRIP orders, and are collections of statements which are executed one after the other when the procedure is called. A procedure is run either by using the **Run** order and the procedure name, or by using the procedure name alone (as long as the name does not conflict with any CCL command).

External host command procedures can also be executed from the CCL command line. However, this section refers exclusively to TRIP procedures which contain TRIP orders and commands.

Procedures currently cannot be run from within TRIPmanager.

Classes of Procedures

There are three classes of procedures: private, group and public.

A *private* procedure can be used only by its owner (creator). To make it available to another user, the owner must **EXPOrt** the procedure to a file and the recipient must **IMPOrt** that file as their own private procedure. If you must use **IMPOrt**/**EXPOrt**, see your User Manager for assistance.

Group procedures are created by the User Manager for the exclusive use of those group members. The procedure names are created from the group name, followed by a period and the name of the procedure. For example:

```
sales.summary
```

would be an appropriate name for a procedure called **Summary**, and is assigned to the user group ‘Sales’.

All TRIP users have access to *public* procedures, which can only be created by the user SYSTEM. These are named using the word ‘Public’, followed by a period and the name of the procedure itself, as in:

```
public.macrohelp
```

Conflicting Names

To explain how procedural name conflicts are resolved, assume that one particular user belongs to two groups, ‘Divisional’ and ‘Sales’, and wants to run a procedure called ‘New_Procedure’. He or she requests that ‘New_Procedure’ be executed by using the **Run** (short form: **R**) command:

```
r new_procedure ↵
```

TRIP searches for ‘New_Procedure’ among this user’s private procedures, and if found, the instructions are carried out. If it is not found, the system looks for procedures named ‘Divisional.New_Procedure’ or ‘Sales.New_Procedure’, those belonging to one of the user’s groups. If any such procedure is found, it is run as above. If there is more than one applicable procedure name, TRIP displays an error message asking for the procedure name in full.

If TRIP cannot find ‘New_Procedure’ among the user’s private or group procedures, it looks for it among the public procedures, and if it finds ‘Public.New_Procedure’, runs it.

Displaying Available Procedures

TRIP is shipped with a variety of public procedures, which are available to all users. To generate a list of these and any others to which you may have access, log on to TRIP and choose **CCL Search**. On the command line, enter the **Show PRocedure** (short form: **S PR**) request

```
s pr ↵
```

A screen similar to this one will appear:

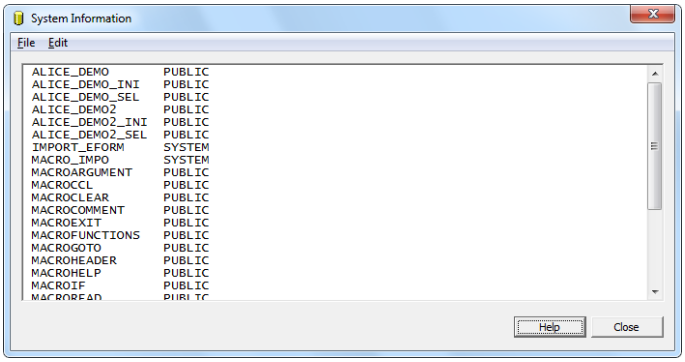

Figure 11–1 The Show Procedure screen

which lists the name of the procedure first, then its class (public, group or private). If you are the creator (owner) of any of these, the class will be 'private' and your username will be displayed in the 'Class' column.

In the sample screen given previously, the procedure called 'A_Sample_Proc' is a private procedure which is owned by user 'Viola'. The remainder are public procedures available to all.

To have this list sent to a printer, use the order **Print PRocedure** (short form: **P PR**):

```
p pr ↵
```

Creating and Modifying Procedures

Procedures are written using an ordinary Edit window. You will need a basic understanding of the system editor and editing commands in use at your installation for these exercises. Consult your system user guide or System Manager for assistance.

Rules for Writing Simple Procedures

Keep the following in mind when writing procedures:

1) Each line may contain one CCL order.
2) A procedure may call another procedure, down to a depth of five levels (procedural *nesting*).
3) Search result numbers used in a procedure are local to that procedure.

We will discuss each of these in later sections.

Creating a Procedure

Note:

Procedure names may contain from one to sixteen alphanumeric characters, and may include the underscore (_).

We will create a simple procedure called ‘Culinary’, which searches for certain foodstuffs mentioned in the demonstration databases.

To create a new procedure, first select the ‘Procedures’ sub tree of ‘My Profile’ in the left hand pane of the TRIPmanager window, then choose ‘New ▸ New Procedure…’ from the ‘Action’ menu, thus:

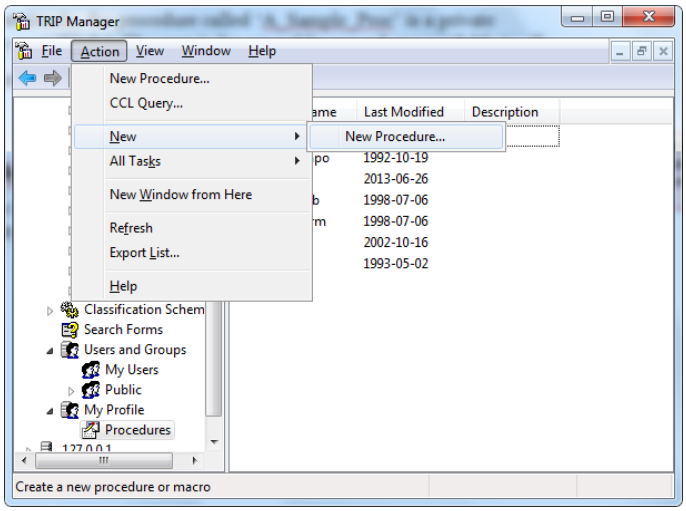

Figure 11–2 The New Procedure… menu

A wizard will now appear, walking you through creating the new procedure:

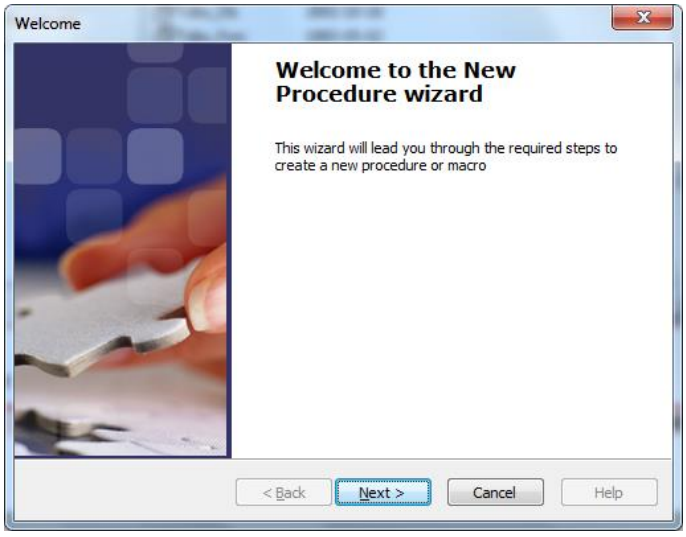

Figure 11–3 The New Procedure wizard

Clicking on the ‘Next’ button will take you to a general details form for entering a name and description for the new procedure; here filled in with information about our procedure, ‘Culinary’.

Enter the details as shown below:

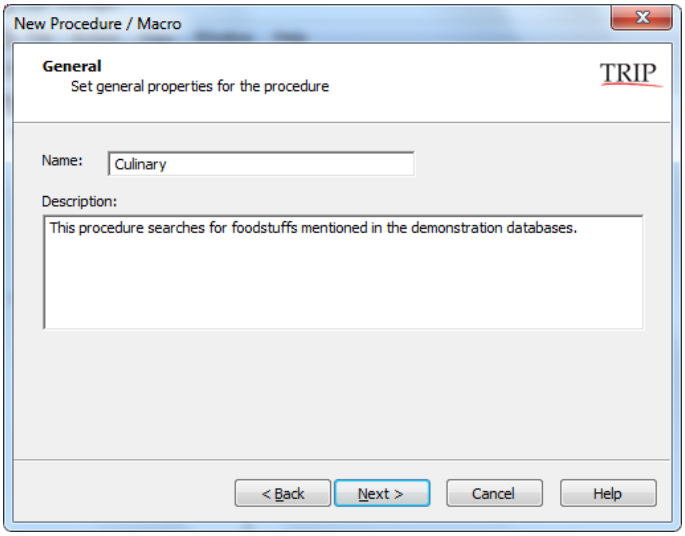

Figure 11–4 Procedure 'Culinary' general properties

Once you have entered your procedure's details, click on the 'Next' button to continue to the procedure content creation form:

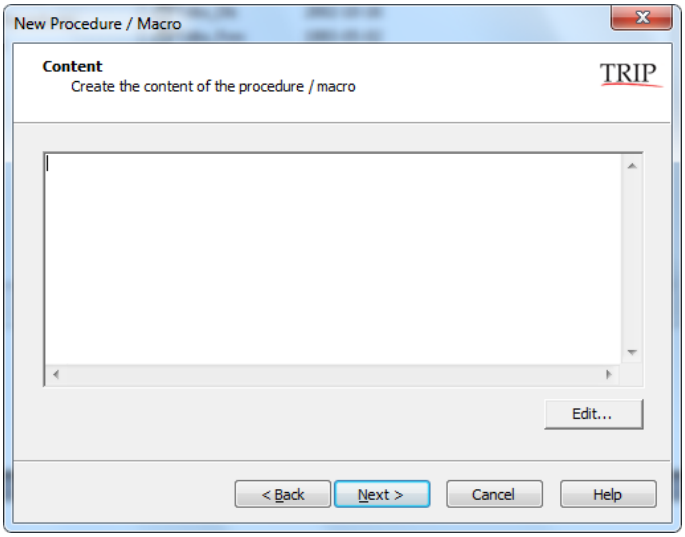

Figure 11–5 The Procedure content form

Here, you can either enter the content of your procedure directly into the content box or, by clicking on the 'Edit' button, you can bring up the default text editor for your installation

Using whichever method you choose to edit the procedure, enter the following line of code:

```
f(cook or food or mutton or pudding) ↵
```

Note:

- *Although the TRIPmanager external text editor is set to use 'Notepad' by default, it is possible to specify your preferred editor. For more information on how to do this, consult the 'TDBS_EDIT' section of this guide.*
- *Only one search order per line (defined by using <Return>) is allowed .*

When you are finished, use the 'Save' and 'Exit' options (on your editor's 'File' menu, if you used an external editor) then click 'Next', or simply click 'Next' (if you entered the commands directly into the content box) and save the series of orders you have entered to the procedure file 'Culinary'.

The procedure content form will now be updated to show the content you have just added to the procedure 'Culinary':

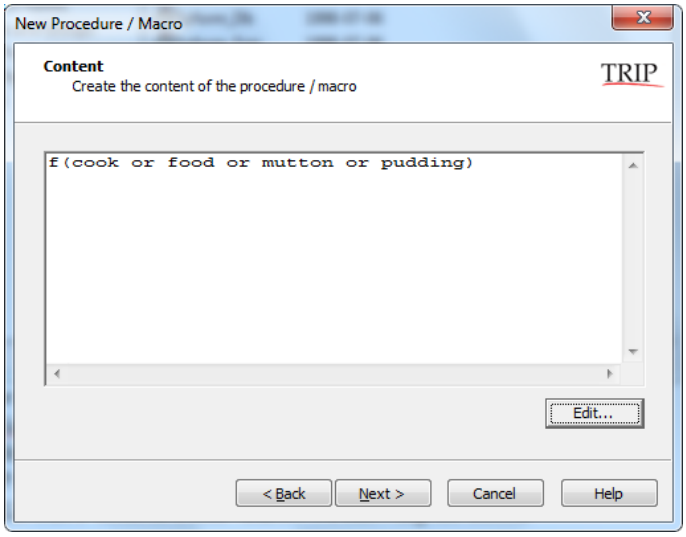

Figure 11–6 Newly added procedure content

Clicking on 'Next' button at this pint will take you to the wizard's finishing screen, where clicking on the 'Finish' button will close the wizard and a confirmation message will be displayed a shown below:

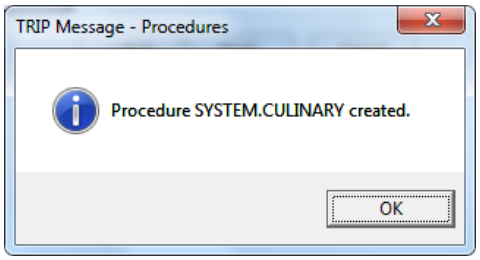

Figure 11–7 Procedure creation success message.

Note:

- *The name of your procedure will automatically be prefixed in dot notation (i.e. author.procedure) with the name of the author (i.e. your username) that created it.*
- *In this case 'author' is the TRIP user 'SYSTEM' and 'procedure' is, of course, 'CULINARY'.*
- *The author and procedure names will only appear as UPPERCASE in the confirmation message.*

Clicking on the 'OK' button will dismiss the message. If the author's name were Viola, for example, TRIP would acknowledges the creation of a new private procedure with the message "Procedure VIOLA.CULINARY created. Press <RET>"

At this point, if you return to the TRIPmanager window, you will be able to see the newly added procedure in the list of procedure available to your user:

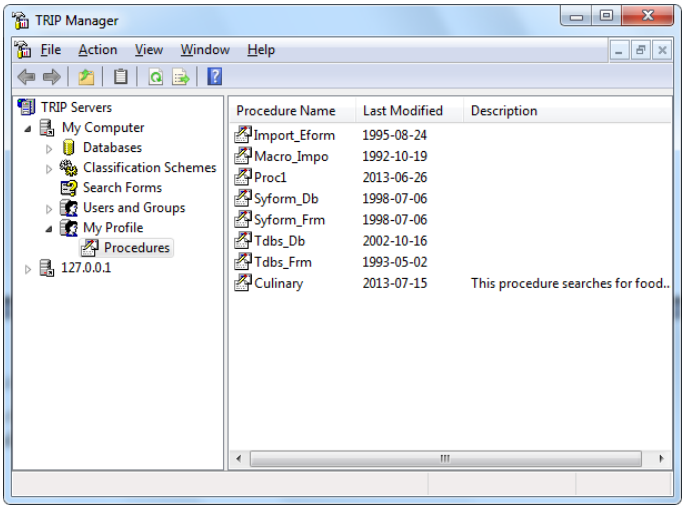

Figure 11–8 List of procedures including 'Culinary'.

To test the new procedure, activate a CCL Query Window on database **Alice** as described in the Introduction of this document, which will produce this statement in the History window:

```
S=1 <475>  BASe alice
```

Execute the procedure from the command entry box:

`r culinary ↵`

or activate the procedure using only its name:

`culinary ↵`

Each command is executed in the order in which it was typed in the procedure,

```
Find (cook OR food OR mutton OR pudding)
```

then logged in the History window as usual:

```
S=2 <20>  Find (cook OR food OR mutton OR pudding)
```

Note:

Execution will stop if one of the orders in the procedure leads to an error and the error will be displayed in a pop-up dialog.

TRIP automatically adjusts the numbering of search results in History when you run a procedure. Run **Culinary** again—the search numbering is automatically incremented by one:

```
S=3 <20>  Find (cook OR food OR mutton OR pudding)
```

Modifying a Procedure

To modify a procedure, first select the 'Procedures' sub tree of 'My Profile' in the left hand pane of the TRIPmanager window, then highlight the desired procedure in the right-hand pane and select 'Properties' from the 'Action' menu:

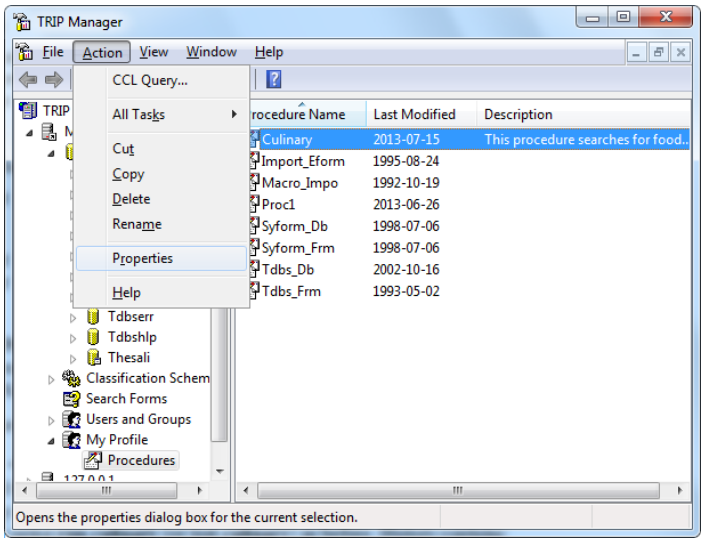

Figure 11–9 Selecting properties for 'Culinary'.

On the properties dialog, select the 'Content' tab and click on the 'Edit' button for the external editor where, as before, the procedure content will appear in the default text editor, or modify the content directly in the 'Content' box:

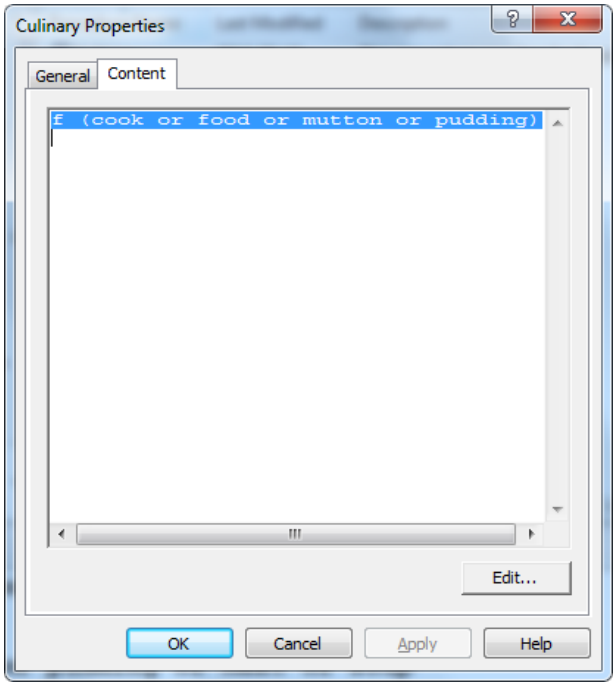

Figure 11–10 Properties 'Content' tab for 'Culinary'.

Until now, the procedure 'culinary' has contained only one instruction:

```
f (cook or food or mutton or pudding)
```

Edit this to read:

```
f (cook or food or mutton or pudding or meat or
soup or
    fish or wine)
```

then save and exit the editor and click 'OK' (if you used an external editor), or click 'OK' if you edited the content directly in the content box..

Note:

At this point, clicking on the 'Apply' instead of clicking 'OK' will store the updated procedure in TRIP but will not quit the 'Properties' dialog, whereas clicking on 'OK' will do both.

Once again, TRIP will show the message "Procedure *author*.CULINARY altered.

Re-open the CCL Query window on database 'Alice' and delete any old CCL orders with:

```
del s=all ↵
```

Although you have deleted the *search statement* that opened database **Alice**, the database remains open, and you may omit the **BASe** command while testing the revised procedure. Activate the edited version using **run culinary** (or just **culinary**) as before. The history now contains:

```
S=1 <37> Find (cook OR food OR mutton OR pudding OR
meat         OR soup OR fish OR wine)
```

with Procedure 'Culinary' examining **Alice** for each of the foodstuffs named above.

Nesting Procedures

Procedural *nesting* involves importing the instructions stored in one procedure into another procedure by using its name in a CCL statement. Nesting can occur to a depth of five levels (that is, one procedure calling another, and that procedure in turn calling yet another) similar to the structure given below:

Procedure One:

CCL statement
CCL statement
Procedure Two
CCL statement
Procedure Three
Procedure Four
CCL statement
Procedure Five
CCL statement
CCL statement
CCL statement
CCL statement
CCL statement
CCL statement.

To create a nested procedure, first create a new a procedure, 'Food_Search', as described in the 'Creating a Procedure' section above, type in the following CCL commands, then save it in TRIP:

```
bas all_bases=alice,corr,thesali
r culinary
```

```
s fo
```

This procedure opens a multifile containing the three databases specified, calls procedure 'Culinary' and searches the multifile for the terms it prescribes, then **Show**s the search result using **FOcus**. When you have successfully created the procedure, you will receive the message "Procedure *author*. FOOD_SEARCH created…" where *author'* is, as always, your username.

Testing the procedure in CCL in the usual way will result in:

```
S=2 <713>    BASe all_bases=alice, corr, thesali
S=3 <37>     Find (cook OR food OR mutton OR pudding
OR meat
                OR soup OR fish OR wine)
```

Copying Procedures

To do this, select the procedures icon under the 'My profile' section of the mmc window, then, in the right hand mmc pane, select the procedure to copy (in this example procedure, 'Fred') and chose 'Copy' from the action menu:

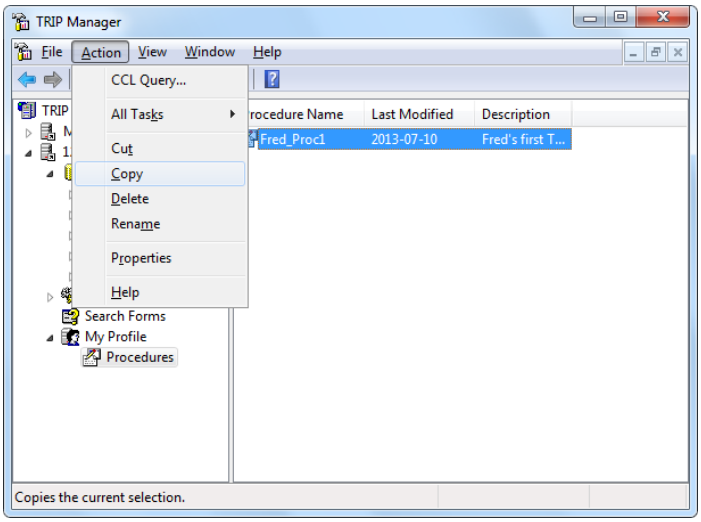

Figure 11–11 Procedure/Macro list screen

Next, deselect the selected procedure and select 'Paste' from the action menu and a renaming box will appear:

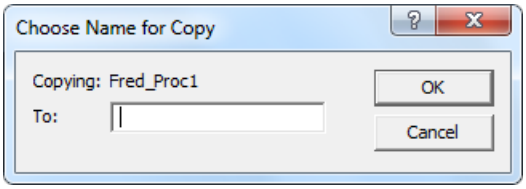

Figure 11–12 Rename Procedure Box

Enter the new name for the copy of the procedure into the 'To:' entry box (in this example the name 'George' has been used) and click on 'OK' to perform the copy. A confirmation box will then appear:

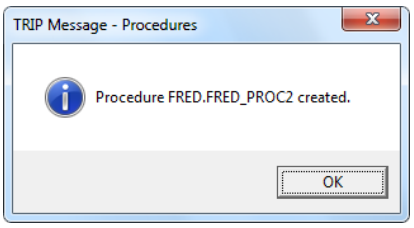

Figure 11–13 Copy Confirmation Box

The procedure has now been copied. In the examples above, the procedure 'Fred', owned by the user 'David' has been copied to a new procedure named, 'George'. The confirmation box confirms the procedure owner and name hence; 'DAVID.GEORGE' is listed as the procedure that was created.

Deleting Procedures

This is achieved by selecting the procedure to delete in the same manner as in a copy action but, this time, selecting 'Delete' from the 'Action' menu:

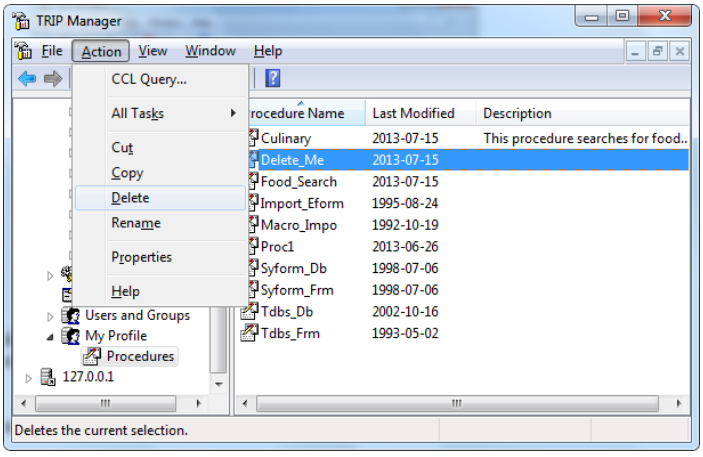

Figure 11–14 Selecting 'Delete' for the procedure 'Delete_me'.

Clicking 'OK' on the confirmation dialogue will delete the procedure. 'Cancel' will abort the operation:

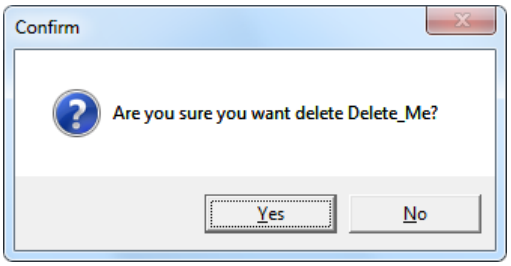

Figure 11–15 Delete confirmation dialog for procedure 'Delete_me'.

As usual, a confirmation dialog will appear on successful deletion:

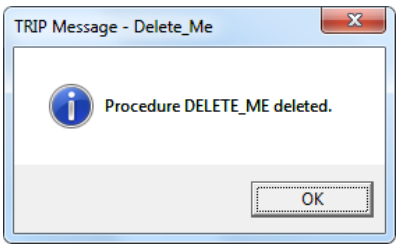

Figure 11–16 Successful procedure deletion message.

and clicking on the 'OK' button will dismiss the success message.

Procedures from SAve Orders

In Chapter Three we discussed storing collections of search results as private procedures using **SAve**, enabling us to copy, edit, delete and embed them *as procedures* in other procedures. To demonstrate, enter **CCL Search** and type in this series of search orders:

```
bas two_databases=alice,corr ↵
f persons ↵
f and mile_high ↵
```

Search History should contain:

```
S=7 <574>  BASe two_databases=alice, corr
S=8 <2>    Find persons
S=9 <1>    Find S=8 AND mile high
```

To store this series as a procedure, use the **SAve searchnumber *procedurename*** command

```
sa s=9 mile_high ↵
```

TRIP confirms with "Search 9 is saved as MILE_HIGH".

Note:

To be able to see the newly created procedure, you must first display the list of procedures by selecting 'My Profile → Procedures' from the left hand pane of TRIPmanager, then select 'Refresh' from the 'Action' menu to update the list.

You can now manipulate this file as you would any other procedure (see Chapter Three for more details).

Note:

Remember that TRIP will not save any CCL statements in the new procedure that are not necessary to produce the final search result. Search results one through six in History will therefore not be included in procedure 'Mile_High'.

Keep in mind that the search result numbers generated within a procedure are *local to that procedure*, and are automatically renumbered to follow whatever sequence is already present in the History window when the procedure is run.

For example, the CCL statements entered on the Command Line in the last example give search results numbered seven, eight and nine:

```
S=7 <574>  BASe two_databases=alice, corr
S=8 <2>    Find persons
S=9 <1>    Find S=8 AND mile high
```

since the previous search registered was result number six. Search result number nine and the orders necessary to produce it were then saved as procedure 'Mile_High'. The instructions now stored *within* 'Mile_High', however, assume internal numbers of their own, and all search orders within that procedure which refer to earlier CCL statements (also internal to that procedure) use these local numbers.

What was previously search result seven in the History window shown above becomes result number one within Mile_High, while result eight becomes two, and nine is renumbered to three:

```
BASe two_databases=alice, corr
Find persons
Find S=2 AND mile high
```

If you run 'Mile_High' again, the search result numbers in History change once more; however, the internal numbering of the procedure remains the same:

```
S=10 <574>  BASe two_databases=alice, corr
S=11 <2>    Find persons
S=12 <1>    Find S=11 AND mile high
```

When writing and testing more complex procedures, it may help to include this **DELete** Search Result request as the first instruction:

```
del s=all ↵
```

This will clear the History window of all previous search results whenever the new procedure is run, and allow accurate tracking of search result numbering during creation and testing. This line can be removed when the procedure has been completed.

To **DELete** procedures created with **SAve**, use one of the methods mentioned previously in 'Deleting Procedures', or try the **DELete SAve** (short form: **DEL SA**) order:

```
del sa procedurename ↵
```

Procedures with Arguments

Making use of TRIP's ability to pass parameters and variables to procedures increases their flexibility. Each piece of data passed is called an *argument*, with the arguments separated by commas, and each might include a date for a search order, a database name or an output format.

A procedure with arguments has two segments, a head and a body. The head contains one line for each parameter passed to the procedure by the user. Up to nine parameters are allowed (**%1** through **%9**), and any mandatory arguments must come first in the procedure head. The other parameters must be given a default value to be used if that argument is omitted when the procedure is used. The procedure body consists of CCL orders, one order on each line.

Try creating a procedure with this set of **Alice** search commands and save it as 'View_Person_Text':

```
%1(?A search expression must follow
View_Person_Text)
%2=person
%3=person, text
bas alice
```

```
f %1
s sor=%2 f=%3
```

Note:

*The text after the query sign (**?**) in the message "(?A search expression…)" must not exceed sixty-eight characters.*

A **Run** order using this procedure in CCL would take the following form:

```
r view_person_text searchterm, fieldname,
formatname
```

At run-time this procedure does the following:

1) finds the procedure named 'View_Person_Text'
2) looks for a mandatory user-provided *search term* (followed by a comma) immediately after the procedure name and stores it in the first argument, **%1**.
   i) It is possible to include customized error message text as part of this statement by typing parentheses, a '**?**' and the desired text (in this example it is "A search expression must follow VIEW_PERSON_TEXT')" which will be displayed if a user tries to run this procedure without the mandatory argument. If no error text is provided, the default error message is "Missing mandatory argument."
3) looks for a user-supplied *field name* (followed by a comma) in which to search for the term stored in **%1** and places in the second argument, **%2**. If the user does not provide a field name, the procedure will use the **person** field by default
4) looks for a user-furnished *format name* with which to display the search result and keeps it in the third argument, **%3**. If none is given, the procedure will use its predefined default, **format=person, text**
5) opens the demonstration database **Alice**
6) **Finds** the search term stored in **%1**
7) and **Shows** the search result **SORt**ed according to the contents of **%2**, and in the **Format** specified by **%3**.

Here are some sample **Run** orders using procedure 'View_Person_Text' with their respective Search Histories:

Example 1:

```
r view_person_text cat,chaptnr,2
S=1 <475>          BASe Alice
S=2 <22>           Find cat
```

shows the output sorted by **chaptnr** and in Format 2.

Example 2:

```
r view_person_text person=rabbit
S=3 <475>    BASe alice
S=4 <34>     Find person=rabbit
```

has the output sorted by **person** and using Format=person, text.

This procedure can be called without the second and/or third argument, in which case the default values given in the procedure are used. If the procedure is called without

the first argument (which is mandatory and for which no default value is given), it results in the error message defined for this parameter in the procedure head being given.

Example 3:

```
r view_person_text white or red,"chaptnr,speaker",1
S=5 <475>    BASe Alice
S=6 <171>    Find white OR red
```

and the output is sorted by **chaptnr** and **speaker**, and shown in Format=1.

As evidenced by the preceding examples, the arguments of a **Run** order are separated by commas. The double quotation marks are necessary, since here the second comma is part of the second argument. Single quotation marks cannot be substituted, as they would be read by TRIP as part of the second argument and would lead to an error message.

Text Substitution Procedures

A special kind of procedure, the *microprocedure*, consists of only a single line of text.

As an example, create a procedure now called 'Alices_Animals', containing this text:

```
(lion or tiger or bear)
```

This procedure may be used in search requests such as:

```
f %alices_animals and soldier# ↵
```

which produces this result:

```
S=7 <2>    Find (lion OR tiger OR bear) AND
soldier#
```

Here you can see that the text of the procedure is inserted instead of the **%** sign and procedure name.

Create another procedure called 'Text_Only', containing this one-line instruction:

```
f=txt,txt2,content sor=person, speaker, rname,
sname
```

This type of procedure can be especially helpful in a search series such as this:

```
bas both=alice,corr ↵
f person ↵
s %text_only
```

where a long string of text is automatically substituted for **%** and a procedure name which can be as short as a single character.

It is also possible to store lists of search terms (one per line) in external files, and incorporate the file names into CCL statements. If the animal terms contained in microprocedure 'Alices_Animals' were saved in an external file called 'More_Animals' in this way:

```
Lion
Tiger
bear
```

and the file name was substituted for the procedure name in the example above, the syntax would be:

```
f file(more_animals.txt) and soldier#
```

with similar results.

Part 5:

Macros

Chapter 12: Macros

Macro Facilities Within TRIP

The macro facilities in TRIP allow users to create their own commands, and make certain extensions to the functionality of the standard TRIP procedures possible.

Creating, modifying, and deleting a macro is done exactly as if it was an ordinary procedure. A macro is invoked for execution by giving a command to TRIP in one of the formats listed below:

```
macroname argument1, argument2...argumentn

%macroname argument1, argument2...argumentn

Run macroname argument1, argument2...argumentn
```

Macro Structure

TRIP macros consist of the following parts:

Macro Header

This part consists of the single reserved word MACRO on the first line of the procedure. It is the macro header that makes TRIP recognize the procedure as a macro procedure and therefore treat it differently than an ordinary procedure.

Argument Specification

This part consists of one or more argument specification statements directly following the macro header, and is equivalent to the procedure head in an ordinary procedure. They must be on separate lines, and can take the following forms:

```
%arg_number

%arg_number=default

%arg_number(?Error_message_text)
```

where:

- Arg_number is a single-digit number from 1 to 9 inclusive,
- default is a value that is to be used as a default (where an optional argument is not specified in the command line that invokes the macro), and
- error_message_text is the text of an error message which is to be displayed on the screen when a required argument is missing from the command line that invokes the macro.

Macro Arguments, Variables and Functions

There are three constructs available which will be assigned values at the time of execution:

Macro Arguments

There are nine arguments available of the form:

```
%n
```

where n is 1, 2, ... 9

In the case of nested macros, the values of all arguments must be given explicitly when the top level macro is invoked. If any of them are used in lower level macros, they must be passed on to the lower level macros by the invoking macro.

Macro Variables

There are twenty-six macro variables, which may be given values through the macro read statements and used in various other macro statements. The form of a macro variable is:

```
%*
```

where * is a letter from the English alphabet, for example

```
%A
```

Macro variables are local to the macro in which they are being used. In the case of nested macros, the values given to the variables in a higher level macro will not change through the execution of a lower level macro.

Macro Functions

The following macro functions are currently available:

Macro Function	Produces
%USER	the current TRIP user
%TIME	the current time
%DATE	the current date
%BASE	the currently-open database
%SEAR	the current search number
%NREC	the number of records in the current search
%NHIT	the number of terms found in the current search
%LANG	the current language
%SUCC	contains ‘Y’ if the most recent CCL order succeeded, and ‘N’ otherwise
%RTNB	contains TRIP’s return code from the preceding CCL order (an odd value implies success)
%RTNA	contains the return value of the most recently called ASE

Table 12–1 Macro functions

Macro Statements

This part consists of one or more macro statements, including CCL, WRITE, READ, LET, CLEAR, IF (conditional), GOTO, COMMENT, TRACK, WAIT or EXIT.

CCL

A CCL statement can be the invocation of a macro or any of the commands currently recognized by TRIP, and will be scanned for references to any of the macro arguments, macro variables, or macro functions. These references will be resolved, and the relevant values substituted in the CCL command.

There is currently a limit of 400 characters on the length of any CCL command, and any CCL command which exceeds this limit after expansion will be rejected as an error.

WRITE

This statement will allow the user to write a text string into the macro window, beginning at a particular row and column. The macro window is nineteen rows deep and seventy-eight columns wide. The text string to be written to the window will be scanned for references to macro arguments, variables, and functions, and these will be resolved when the string is written. The format of the statement is:

```
WRITE(row, column, 'text_string')
```

Any row or column value outside the permitted range will be reflected as an error. A text string which extends beyond the maximum column number will cause the macro to terminate with an error message.

Note:

By default the contents of the macro window are presented in 'Bold'. To cancel this, enter the statement:

```
WRITE(normal)
```

before the first WRITE statement of the macro.

READ

This statement will allow TRIP to read a text string from the keyboard into a macro variable. The format of the statement is:

```
READ(row, column, 'prompt_text',
        macro_variable)
```

where:

- *Row* and *column* are the row and column respectively where the *prompt_text* will start,
- *prompt_text* is the text to be used as a prompt for the read operation, and
- *macro_variable* is the variable to which the input string is to be assigned.

LET

This statement is an alternative way of putting a text string into a macro variable. The format of the statement is:

```
LET macro_variable=text_string
```

A macro function may be inserted in the variable in the same fashion.

CLEAR

This allows the macro window or a number of lines in it to be cleared within a MACRO. The format of the statement is:

```
CLEAR(first_row, last_row)
```

where:

first_row is the first row to be cleared, and

last_row is the last row to be cleared.

IF

This statement will allow the user to test on various conditions and control the execution of statements within the macro. Each statement must start on a new line, and IF statements can be nested.

The formats of the IF ...statement are:

```
IF condition THEN
     statement (s)
ENDIF
```

or

```
IF condition THEN
     statement1 (s)
ELSE
     statement2 (s)
ENDIF
```

The format of conditions is '*expression operator expression*', and the formats of expression are '*macro argument*/*macro variable*/ *macro function*/*list of constants*', where a constant is a string of characters, and each element of the list is separated by a comma.

The formats of operator are:

Operator	Function
EQ	equal to
GT	greater than
LT	less than
GE	greater than or equal to
LE	less than or equal to
NE	not equal to
IN	contained in

Table 12–2 IF Statement Conditional Operators

The IN operator can only be used when the expression following it consists of a list of more than one constant, for example

```
IF %A IN y,n THEN
        …
ENDIF
```

The formation of statement(s), statement1(s), and statement2(s) is any macro statement.

GOTO

This statement will allow the user to specify a line in the macro to which flow of control will then go. The format of the GOTO statement is:

```
GOTO label
```

where *label* is a character string

Somewhere in the macro there must be a line similar to this:

```
- label
```

and execution will resume with the line following this one.

COMMENT

Any line beginning with a single asterisk [*] or exclamation point [!] in a macro will be treated as a comment line, and no further interpretation of it will be given when the macro is executed.

This is an example of a two-line comment:

```
* This is a comment line, followed immediately * by
another such comment line!
```

TRACK

This displays the macro commands in the TRIP command window as they are being executed, enabling the user to follow the flow of control as the macro is running. The format of the TRACK statement is:

```
TRACK n
```

where *n* is the number of seconds that the command is to be displayed before it is actually executed.

WAIT

This suspends execution of the macro for the specified number of seconds. The format of the statement is:

```
WAIT n
```

where *n* is the number of seconds the execution will be delayed before resuming at the next line.

EXIT

If you wish a macro to return to its point of activation before executing its last line, use the EXIT command. For example,

```
IF %A EQ 1 THEN

ELSE

ENDIF
```

Macro Examples

When your TRIP system is delivered, an online help facility gives the following information in menu form. These menus and the texts they present are themselves contained in macros. Macro help is invoked by activating the macro MACROHELP from the CCL command line:

```
macrohelp
```

The menu macros may serve as examples of how to create macros, and are reproduced here for reference.

PUBLIC.MACROHELP

```
MACRO
*
- start
WRITE ( 4,10,' Introduction to the MACRO facilities within TRIP.
')
WRITE ( 6,10,' The MACRO facilities in TRIP allow for certain
extensions  ')
WRITE ( 7,10,' to the functionality of the standard TRIP
procedures.  ')
WRITE (12,10,' Choose one of the following options  :--  ')
WRITE (14,10,' 1 The Structure of a macro  ')
WRITE (15,10,' 2 MACRO Statements  ')
WRITE (16,10,' 3 MACRO Functions  ')
WRITE (17,10,' 4 Quit this Introduction  ')
READ (19,30,'CHOICE  : ',%A)
*
IF %A EQ 1 THEN
   %PUBLIC.MacroStructure
ELSE
   IF %A EQ 2 THEN
        %PUBLIC.MacroStatements
   ELSE
      IF %A EQ 3 THEN
          %PUBLIC.MacroFunctions
      ELSE
         IF %A EQ 4 THEN
            GOTO stop
         ENDIF
      ENDIF
   ENDIF
ENDIF
GOTO start
*
- stop
EXIT
```

PUBLIC.MACROSTRUCTURE

```
MACRO
*
- start
WRITE ( 3, 2,'The Structure of TRIP Macros.  ')
WRITE ( 5, 2,'TRIP Macros consist of the following parts  :-- ')
WRITE ( 7, 2,' 1 The MACRO header.  ')
WRITE ( 8, 2,' 2 The Argument specification section.  ')
WRITE ( 9, 2,' 3 The Macro statement section.  ')
WRITE (10, 2,' 4 Quit.  ')
READ (19,30,'CHOICE  : ',%A)
*
IF %A EQ 1 THEN
   %PUBLIC.MacroHeader
ELSE
   IF %A EQ 2 THEN
      %PUBLIC.MacroArgument
   ELSE
      IF %A EQ 3 THEN
        %PUBLIC.MacroStatementSe
      ELSE
         IF %A EQ 4 THEN
            GOTO stop
         ENDIF
      ENDIF
   ENDIF
ENDIF
GOTO start
*
- stop*
*
EXIT
```

PUBLIC.MACROHEADER

```
MACRO
*
WRITE ( 4,20,' THE MACRO HEADER')
WRITE ( 8,10,' This part consists of the single reserved word
MACRO')
WRITE ( 9,10,' on a line. ')
READ (19,30,' Press <CR> ',%A)
EXIT
```

PUBLIC.MACROARGUMENT

```
MACRO
*
WRITE ( 3,20,'THE MACRO ARGUMENT SPECIFICATION SECTION')
WRITE ( 4,10,'This part consists of one or more argument
specification')
WRITE ( 5,10,'statements directly following the MACRO header. They')
WRITE ( 6,10,'must be on separate lines and can have the following
forms  :-')
WRITE ( 8,10,' a. %Arg_Number')
WRITE ( 9,10,' b. %Arg_Number=Default')
WRITE (10,10,' c. %Arg_Number(?Error_message_text)')
WRITE (12,10,'where Arg_Number is a single digit number from 1 to 9
inclusive,')
WRITE (13,10,'Default is a value that is to be used as a default in the
case')
WRITE (14,10,'where an optional argument is not specified in the
command line')
WRITE (15,10,'that invokes the macro, and Error_message_text is the
text of')
WRITE (16,10,'an error message which is to be displayed on the screen
when a')
WRITE (17,10,'required argument is missing from the command line that')
WRITE (18,10,'invokes the macro.')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROSTATEMENTSE

```
MACRO
*
WRITE ( 3,20,'The MACRO STATEMENT SECTION')
WRITE ( 5,10,'This part consists of one or more macro statements.
These')
WRITE ( 6,10,'statements can be of one of the following types:')
WRITE ( 8,15,' CCL. ')
WRITE ( 9,15,' Conditional.')
WRITE (10,15,' GOTO.')
WRITE (11,15,' Write.')
WRITE (12,15,' Read.')
WRITE (16,10,'These statements are documented in more detail in
the MACRO')
WRITE (17,10,'statements option.')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROSTATEMENTS

```
MACRO
*
- start
WRITE ( 3,2,' Macro Statements.')
WRITE ( 5,2,' 1 WRITE')
WRITE ( 6,2,' 2 READ')
WRITE ( 7,2,' 3 IF...')
WRITE ( 8,2,' 4 GOTO')
WRITE ( 9,2,' 5 EXIT')
WRITE (10,2,' 6 CCL command')
WRITE (11,2,' 7 TRACK')
WRITE (12,2,' 8 COMMENT')
WRITE (13,2,' 9 CLEAR')
WRITE (15,2,' 10 Quit.')
READ (18,10,'CHOICE  : ',%A)
*
IF %A EQ 1 THEN
   %PUBLIC.MacroWRITE
ELSE
   IF %A EQ 2 THEN
       %PUBLIC.MacroREAD
   ELSE
      IF %A EQ 3 THEN
         %PUBLIC.MacroIF
      ELSE
         IF %A EQ 4 THEN
             %PUBLIC.MacroGOTO
         ELSE
            IF %A EQ 5 THEN
                %PUBLIC.MacroEXIT
            ELSE
               IF %A EQ 6 THEN
                   %PUBLIC.MacroCCL
               ELSE
                  IF %A EQ 7 THEN
                      %PUBLIC.MacroTRACK
                  ELSE
                     IF %A EQ 8 THEN
                          %PUBLIC.MacroCOMMENT
                     ELSE
                        IF %A EQ 9 THEN
                            %PUBLIC.MacroCLEAR
                        ELSE
                           IF %A EQ 10 THEN
                           GOTO stop
                           ENDIF
                        ENDIF
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF
ENDIF
GOTO start
*
- stop
EXIT
```

PUBLIC.MACROWRITE

```
MACRO
WRITE ( 3,2,'The Macro WRITE Statement. ')
WRITE ( 5,2,'This statement will allow a user to write a text
string into the')
WRITE ( 6,2,'Macro window beginning at a particular Row and
Column. The Macro')
WRITE ( 7,2,'window is 19 Rows deep and 78 Columns wide. The text
string to be')
WRITE ( 8,2,'written to the window will be scanned for references
to Macro ')
WRITE ( 9,2,'arguments variables and functions and these will be
resolved ')
WRITE (10,2,'the string is written. The format of the statement
is :-- ')
WRITE (12,2,' WRITE (Row,Column,Text_String) ')
WRITE (14,2,'Any Row or Column Value outside the permitted range
will be ')
WRITE (15,2,'errored. A text string which extends beyond the
maximum ')
WRITE (16,2,'column number will be truncated.')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROREAD

```
MACRO
*
WRITE ( 4,20,'The READ STATEMENT')
WRITE ( 6,10,'This statement will allow TRIP to read a text
string from')
WRITE ( 7,10,'the macro window into a macro variable.')
WRITE ( 9,10,'Form of the READ statement.')
WRITE (11,10,'READ (Row,Column,Prompt_text,macro_variable)')
WRITE (13,10,'where Row and Column are the Row and Column
respectively where')
WRITE (14,10,'the Prompt_text will start; Prompt_text is the text
to be used')
WRITE (15,10,'as a prompt for the read operation, and
macro_variable is the ')
WRITE (16,10,'variable to which the input string is to be
assigned.')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROIF

```
MACRO
*
WRITE ( 4,20,'The IF .... STATEMENT.')
WRITE ( 6,10,'This statement will allow the user to test on
various')
WRITE ( 7,10,'conditions and control the execution of
statements')
WRITE ( 8,10,'within the macro.')
WRITE (10,10,'a. Forms of the Conditional statement.')
WRITE (12,10,' i. IF Condition THEN ii. IF Condition THEN')
WRITE (13,10,' Statement[s]  Statement1[s]')
WRITE (14,10,' ENDIF ELSE')
WRITE (15,10,' Statement2[s]')
WRITE (16,10,' ENDIF')
READ (19,30,'Press <CR> for more information ....',%A)
CLEAR (10,19)
WRITE (10,10,'b. Form of the Condition.')
WRITE (12,10,' i. Expression Operator Expression')
READ (18,20,'Press <CR> for more information .... ',%A)
CLEAR (10,19)
WRITE (10,10,'c. Forms of the Expression.')
WRITE (12,10,' i. macro argument ')
WRITE (13,10,' ii. macro function ')
WRITE (14,10,' iii. macro variable ')
WRITE (15,10,' iv. list of contents')
WRITE (16,10,'where the constant is a string of characters, and
elements of ')
WRITE (17,10,'the list are separated by a comma.')
READ (19,30,'Press <CR> for mre information .... ',%A)
CLEAR (10,19)
WRITE (10,10,'d. Forms of the Operator.')
WRITE (12,10,' i. EQ Equal to iv. LE Less than or Equal to')
WRITE (13,10,' ii. GE Greater than or Equal to v. LT Less Than')
WRITE (14,10,'iii. GT Greater Than vi. IN contained IN')
WRITE (16,10,'where the IN operator can only be used in the case
where ')
WRITE (17,10,'Expression is a list of more than one constant.')
READ (19,30,'Press <CR> for more information .... ',%A
CLEAR (10,19)
WRITE (10,10,'e. Form of the statement.')
WRITE (12,10,'Any macro statement.')
READ (19,30,'Press <CR> to continue .... ',%A)
EXIT
```

PUBLIC.MACROGOTO

```
MACRO
*
WRITE ( 4,20,'THE GOTO STATEMENT')
WRITE ( 6,10,'This statement will allow the user to specify a
line in the ')
WRITE ( 7,10,'macro to which flow of control will then go.')
WRITE ( 9,10,'Form of the GOTO statement.')
WRITE (11,10,'GOTO label')
WRITE (13,10,'where the line being specified is preceded by a
label. A ')
WRITE (14,10,'label must be in the format "- labelname".')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROEXIT

```
MACRO
*
WRITE ( 4,10,'THE EXIT STATEMENT')
WRITE (10,10,'Every MACRO must end with the single reserved word
"EXIT"')
WRITE (11,10,'It appears as the last statement in the macro.')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROCCL

```
MACRO
*
WRITE ( 6,20,'A CCL STATEMENT.')
WRITE ( 8,10,'A CCL statement can be any of the commands
currently')
WRITE ( 9,10,'recognised by TRIP. A CCL statement will be
scanned')
WRITE (10,10,'for references to any of the macro arguments, macro
')
WRITE (11,10,'variables, or macro functions, and these references
will')
WRITE (12,10,'be resolved, and the relevant values substituted in
the ')
WRITE (13,10,'CCL command. Note that there is currently a limit
of 400')
WRITE (14,10,'on the length of any CCL command, and any CCL
command which')
WRITE (15,10,'excedes this limit will be errored. ')
READ (19,30,'Press <CR> ',%A)
EXIT
```

PUBLIC.MACROTRACK

```
MACRO
*
WRITE ( 4,20,'THE TRACK COMMAND')
WRITE ( 6,10,'This displays the commands in the MACRO onto the
screen')
WRITE ( 7,10,'as they are being executed. You can then follow the
flow')
WRITE ( 8,10,'of control as the MACRO is running.')
WRITE (10,10,'Form of the statement;-')
WRITE (12,10,'TRACK n')
WRITE (14,10,'where n is the number of seconds that the command
is to')
WRITE (15,10,'be displayed before it is actually executed.')
READ (19,30,'Press <RET> ',%A)
*
EXIT
```

PUBLIC.MACROCOMMENT

```
MACRO
*
WRITE ( 4,20,'THE COMMENT STATEMENT')
WRITE ( 8,10,'This consists of a single asterisk (*) at the
beginning of')
WRITE ( 9,10,'any line within the MACRO. Anything following the
asterisk')
WRITE (10,10,'is then ignored.')
WRITE (12,10,'Form of the statement ;-')
WRITE (14,10,'* This is a comment line ')
READ (19,30,'Press <RET> ',%A)
*
EXIT
```

PUBLIC.MACROCLEAR

```
MACRO
*
WRITE ( 4,20,'THE CLEAR COMMAND')
WRITE ( 8,10,'This allows the screen or part of the screen to be
cleared')
WRITE ( 9,10,'within a MACRO.')
WRITE (11,10,'Form of the statement;-')
WRITE (13,10,'CLEAR (first_row,last_row)')
WRITE (15,10,'where first_row is the first row to be cleared and
last_row')
WRITE (16,10,'is the last row to be cleared.')
READ (19,30,'Press <RET>',%A)
*
EXIT
```

PUBLIC.MACROFUNCTIONS

```
MACRO
*
* This is a demonstration Macro to display the
* MACRO functions that are available at present.
*
WRITE  ( 3,10,'The following MACRO functions are currently
available :--')
WRITE  ( 5,3,'USER   The current TRIP user is %USER
')
WRITE  (6,3,'TIME   The current Time is %TIME
')
WRITE  (7,3,'DATE   The current Date is %DATE
')
WRITE  (8,3,'BASE   The currently open Database is %BASE
')
WRITE  (9,3,'SEAR   The current Search Number is %SEAR
')
WRITE  (10,3,'NREC   The number of Records in the current search is
%NREC      ')
WRITE  (11,3,'NHIT   The number of Terms found in the current
search is %NHIT  ')
WRITE  (12,3,'LANG   The current CCL-Language is %LANG
')
WRITE  (13,3,'SUCC   The letter indicating the Success of the most
recently   ')
WRITE  (14,3,'       executed CCL-command is %SUCC   ')
WRITE  (15,3,'RTNB   The Return Code from the most recent CCL-
command is %RTNB ')
WRITE  (16,3,'RTNA   The Return Code from the ASE most recently
called from -
 CCL is %RTNA')
READ   (19,30,'Press <CR> ',%A)
EXIT
```

Part 6:

Appendices

Appendix A: TRIPsystem CLI Switches

While not pertaining to managing TRIPsystem using TRIPmanager, CLI (Command Language Interpreter) switches are nonetheless useful for performing actions from the system prompt to facilitate many processes in TRIP, including logging into TRIPclassic with direct access to data entry and search forms and CCL.

CLI switches can also give access to the TRIPsystem version number and some language control functions normally associated with terminal sessions.

The CLI switches are outlined in the table below, with the shortest form permitted for each (if available) indicated by the upper-case, bolded letters.

Switch	Legal Values	Function
-v	none	Displays the TRIPsystem version number
-u *username*	any valid TRIP user name	entry of TRIP username
-p *password*	any valid TRIP password	entry of TRIP user password
-t *terminal type*	TTY, VT100, VT200 and site-dependent	specification of terminal type
-s **C**cl	**C**cl	specifies direct entry into the CCL command window
-C ***'CCL command'***	any valid CCL command, procedure or macro	Begins the CCL session with the execution of the order or TRIP procedure specified by *CCL command.* Single quotes are necessary only if the CCL order to be run contains spaces.
-q	none	*R*uns a procedure without echoing the executed commands to the CCL window when used in conjunction with the -C ***'CCL command'*** switch.
-s **F**orms	**F**orms	specifies direct entry into forms searching
-S *formname*	any valid search form name	specifies direct entry to forms search, using the search form named
-o	none	when used in conjunction with the CLI search switches, restricts the user to that portion of TRIP to which he or she has been directed, i.e. CCL, search forms or data entry
-e ***option***	**A**dd, **M**odify or **D**elete	specifies direct access to the desired data entry option using the values **A**dd, **M**odify or **D**elete (the default is **A**dd)
-e *option database_name*	**A**dd, **M**odify or **D**elete and any valid database name	specifies direct access to the desired data entry option and database using the options **A**dd, **M**odify or **D**elete and the database name provided. If a default data entry form has not been defined for the database named, the user will be prompted for an entry form name.
-c *character set*	**CHI**, **DAN**, **ENG**, **FIN**, **GER**, **LA1**, **LA2**, **LA3**, **MUL**, **NOR**, **ROM**, **SWE**, **YUG**	specifies a TRIP national character set (needed only when running TRIPclassic from a 7-bit terminal and using a non-English character set). The default is **MUL**tinational.
-l *language*	**CHI**, **DAN**, **ENG**, **FIN**, **FRE**, **GER**, **ITA**, **NOR**, **SWE**	designates a national variant of the command language CCL with which the user speaks to TRIP and TRIP to the user. The ANSI/CCL standard is **ENG**lish.
-L *language*	**CHI**, **DAN**, **ENG**, **FIN**, **FRE**, **GER**, **ITA**, **NOR**, **SWE**	specifies a national variant of the command language CCL with which the user speaks to TRIP, and will override any user-to-TRIP setting defined with the /**LA**nguage or -l switch.
-f *filename*	any valid filename	creates a log file of the CCL orders given in a particular session. If no file name is specified, the log file will be named trip.log.

Table A–1 CLI Switches

CLI usage Examples

To start TRIP using one or more UNIX/Windows CLI switches, type

```
trip
```

and the switch or switch combination at the system prompt, then press ↵.

Here are three examples:

Example 1:

```
trip -u danny -p ad -f try.log -s ccl
```

starts TRIPclassic for user ‘Danny’, whose TRIP password is ‘ad’ and whose CCL orders will be written to a log file named ‘try.log’, and who will enter the CCL command window directly.

Example 2:

```
trip -u john -o -s ccl -C john1
```

starts TRIPclassic for user ‘John’, will send him directly to the command window, limit him to *only* that window and execute the procedure called ‘john1’.

Example 3:

```
trip -L eng -l swe
```

starts TRIPclassic with all menus and text in Swedish and all CCL commands in English.

Part 6:

Lists and Index

List of Figures

List of Tables

Index