



digital  
vision  
group

## **TRIPsystem**

CCL Command Reference

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

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

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



## Table of Contents

<b>CONVENTIONS USED IN THIS REFERENCE .....</b>	<b>8</b>
<b>CCL COMMAND REFERENCE – PART 1:.....</b>	<b>9</b>
THE INTERFACES.....	9
<i>Introduction .....</i>	<i>9</i>
<i>CCL Command reference - Part one: The interfaces .....</i>	<i>9</i>
<i>CCL Command reference - Part two: The commands .....</i>	<i>9</i>
STRUCTURE OF A CCL ORDER .....	10
<i>Overview .....</i>	<i>10</i>
<i>Maximum number of characters in a CCL Order.....</i>	<i>10</i>
CCL AND TRIPMANAGER .....	11
CCL AND TRIPCLASSIC .....	12
<i>Opening a TRIPclassic CCL Command Window .....</i>	<i>12</i>
<i>CCL and the TRIPclassic Numeric Keypad.....</i>	<i>13</i>
<b>CCL COMMAND REFERENCE - PART 2:.....</b>	<b>14</b>
THE COMMANDS.....	14
<i>Back .....</i>	<i>14</i>
<i>BASE.....</i>	<i>15</i>
<i>CAL.....</i>	<i>17</i>
<i>CLOSE.....</i>	<i>18</i>
<i>Continue.....</i>	<i>19</i>
<i>DEfine .....</i>	<i>21</i>
<i>DEfine – Section Index .....</i>	<i>21</i>
<i>DEfine – A Simple Order.....</i>	<i>23</i>
<i>DEfine , (Comma) .....</i>	<i>24</i>
<i>DEfine ABout .....</i>	<i>26</i>
<i>DEfine AND.....</i>	<i>27</i>
<i>DEfine CENTury MINimum.....</i>	<i>29</i>
<i>DEfine COMmand KEY (TRIPclassic only) .....</i>	<i>30</i>
<i>DEfine Continue.....</i>	<i>31</i>
<i>DEfine COPY .....</i>	<i>32</i>
<i>DEfine CUT.....</i>	<i>33</i>
<i>DEfine Delete .....</i>	<i>34</i>
<i>DEfine Display .....</i>	<i>35</i>
<i>DEfine EForm (TRIPclasic only) .....</i>	<i>36</i>
<i>DEfine Find (FUZZ).....</i>	<i>37</i>
<i>DEfine Find (Max/Min) .....</i>	<i>38</i>
<i>DEfine FOCUS .....</i>	<i>39</i>
<i>DEfine Format.....</i>	<i>40</i>
<i>DEfine FUZZ .....</i>	<i>42</i>
<i>DEfine Hlghlight.....</i>	<i>44</i>
<i>DEfine HOLD.....</i>	<i>46</i>
<i>DEfine KEY (TRIPclassic only).....</i>	<i>47</i>
<i>DEfine KVP.....</i>	<i>48</i>
<i>DEfine LPcode.....</i>	<i>49</i>
<i>DEfine MAP .....</i>	<i>50</i>
<i>DEfine MASK.....</i>	<i>52</i>
<i>DEfine MAXimum/MINimum .....</i>	<i>55</i>
<i>DEfine MERGe .....</i>	<i>57</i>
<i>DEfine Page (TRIPclassic only) .....</i>	<i>58</i>
<i>DEfine PCode .....</i>	<i>59</i>
<i>DEfine Print.....</i>	<i>60</i>
<i>DEfine PRINTER (UNIX only).....</i>	<i>61</i>
<i>DEfine REVERSE.....</i>	<i>62</i>
<i>DEfine SAvE BASE .....</i>	<i>64</i>
<i>DEfine SCope .....</i>	<i>65</i>

## Table of Contents

<i>DEfine SCoPe SDI</i> .....	66
<i>DEfine SORT</i> .....	68
<i>DEfine SPace</i> .....	69
<i>DEfine – STEMming</i> .....	73
<i>DEfine STop WOrd</i> .....	74
<i>DEfine THESaurus</i> .....	75
<i>DEfine TIMEForm</i> .....	77
<i>DEfine TODay EXPand</i> .....	78
<i>DEfine TStamp</i> .....	79
<i>DEfine UNKNowN FieLd</i> .....	80
<i>DEfine View</i> .....	81
<i>DEfine WEIght</i> .....	83
<i>DElete</i> .....	85
<i>DElete – Section Index</i> .....	85
<i>DElete – Search Result</i> .....	87
<i>DElete – Print Request</i> .....	88
<i>DElete – Procedure</i> .....	89
<i>DElete – Global Update</i> .....	90
<i>Display</i> .....	92
<i>Display – Section Index</i> .....	92
<i>Display – The TRIPmanager Display List</i> .....	94
<i>Display – The TRIPclassic Display List</i> .....	96
<i>Display – A Simple Order</i> .....	98
<i>Display – Field Qualifiers</i> .....	99
<i>Display BASE</i> .....	101
<i>Display CLASS()</i> .....	102
<i>Display – Enhanced Output</i> .....	103
<i>Display – FRequency</i> .....	104
<i>Display FUZZ()</i> .....	105
<i>Display – Search Results</i> .....	107
<i>Display – Sort in REVerse order</i> .....	108
<i>Display – Sort on FRequency</i> .....	109
<i>Display – STEMming</i> .....	110
<i>Display – Thesaurus Modifiers</i> .....	111
<i>Edit (TRIPclassic only)</i> .....	114
<i>Edit – Section Index</i> .....	114
<i>Edit – A Simple Order</i> .....	116
<i>Edit – Search Result</i> .....	117
<i>Edit – INSert</i> .....	118
<i>Edit – PART</i> .....	119
<i>Edit – COPy</i> .....	120
<i>Edit – SORT</i> .....	121
<i>Expand</i> .....	122
<i>EXPORT</i> .....	123
<i>Find</i> .....	125
<i>Find – Section Index</i> .....	126
<i>Find – A Simple Order</i> .....	127
<i>Find – AND operator</i> .....	129
<i>Find – OR operator</i> .....	131
<i>Find – XOR operator</i> .....	132
<i>Find – NOT operator</i> .....	133
<i>Find within fields</i> .....	135
<i>Find if Field Content Exists</i> .....	136
<i>Find – Truncation and Masking</i> .....	137
<i>Find – Proximity Searching</i> .....	140
<i>Find – Previous Searches</i> .....	143
<i>Find Terms From a Display List</i> .....	145
<i>Find – Order of Operator Precedence</i> .....	147

## Table of Contents

<i>Find – FREquency</i> .....	148
<i>Find FUZZ</i> .....	149
<i>Find – Relational Operators</i> .....	151
<i>Find Record number</i> .....	155
<i>Find SAVE</i> .....	157
<i>Find TStamp</i> .....	158
<i>Find TODAY</i> .....	159
<i>Find DUPLICATE</i> .....	161
<i>Find USer</i> .....	162
<i>Find GROUP</i> .....	163
<i>Find THESaurus</i> .....	164
<i>Find – Mapped Transaction Sets</i> .....	166
<i>Find – Internal Transaction Sets</i> .....	168
<i>Find – External Transaction Sets</i> .....	169
<i>Find ABout</i> .....	171
<i>Find Class</i> .....	172
<i>Find SCoPe</i> .....	173
<i>FRequency</i> .....	174
<i>Frequency - Date/Time</i> .....	176
<i>FUZZ</i> .....	177
<i>Help</i> .....	180
<i>HIDe (TRIPclassic only)</i> .....	182
<i>IMPOrt</i> .....	183
<i>INDex</i> .....	185
<i>INSert</i> .....	186
<i>LEAve (TRIPclassic only)</i> .....	189
<i>LIsT</i> .....	190
<i>LOAd</i> .....	191
<i>MEasure</i> .....	192
<i>More</i> .....	193
<i>Next</i> .....	194
<i>PREVious</i> .....	195
<i>Print</i> .....	196
<i>Print – Section Index</i> .....	196
<i>Print – A Simple Order</i> .....	199
<i>Print ACcess</i> .....	200
<i>Print BASE</i> .....	201
<i>Print Display</i> .....	203
<i>Print EForm</i> .....	204
<i>Print FILE</i> .....	205
<i>Print FOCUS</i> .....	207
<i>Print Format</i> .....	208
<i>Print FREquency</i> .....	210
<i>Print GRoup</i> .....	211
<i>Print Highlight</i> .....	212
<i>Print LIst</i> .....	213
<i>Print Local (TRIPclassic only)</i> .....	214
<i>Print MEasure</i> .....	216
<i>Print PART SORT</i> .....	217
<i>Print PRINTER</i> .....	218
<i>Print PRocedure</i> .....	219
<i>Print – Record</i> .....	220
<i>Print REVerse</i> .....	221
<i>Print – Search Result</i> .....	222
<i>Print SForm</i> .....	223
<i>Print SORT</i> .....	224
<i>Print TForm</i> .....	226
<i>Print – Timing</i> .....	228

## Table of Contents

<i>Print TRace</i> .....	230
<i>Print USer</i> .....	231
<i>RENumber</i> .....	232
<i>REVeal (TRIPclassic only)</i> .....	233
<i>Run</i> .....	234
<i>SAve</i> .....	235
<i>Select (TRIPclassic only)</i> .....	237
<i>SForm</i> .....	239
<i>Show</i> .....	240
<i>Show – Section Index</i> .....	240
<i>Show – A Simple Order</i> .....	242
<i>Show ACcess</i> .....	243
<i>Show BASE</i> .....	244
<i>Show COST</i> .....	246
<i>Show EForm</i> .....	247
<i>Show FOCus</i> .....	248
<i>Show Format</i> .....	249
<i>Show GRoup</i> .....	252
<i>Show Hlghlight</i> .....	253
<i>Show PART SORT</i> .....	254
<i>Show PROcedure</i> .....	255
<i>Show – Record</i> .....	257
<i>Show REVerse</i> .....	259
<i>Show – Search Result</i> .....	260
<i>Show SForm</i> .....	261
<i>Show SORT</i> .....	262
<i>Show USer</i> .....	264
<i>STatus</i> .....	265
<i>STOP</i> .....	268
<i>Top</i> .....	269
<i>TRace</i> .....	270
<i>UPDate</i> .....	271
<i>UPDate – Section Index</i> .....	271
<i>UPDate – Global Updating</i> .....	272
<i>UPDate SCope</i> .....	275
<i>UPDate SDI</i> .....	276
<b>LIST OF FIGURES</b> .....	<b>277</b>
<b>LIST OF TABLES</b> .....	<b>277</b>
<b>START OF MAIN INDEX</b> .....	<b>278</b>



## Conventions Used in this Reference

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

<i>italic</i>	Used to indicate variables such as <i>fieldtype</i> or <i>databasename</i> , and to emphasize important <i>information</i> , <i>terms</i> and <i>concepts</i>
<b>bold</b>	Used to indicate anything that TRIP recognizes or can interpret and act upon, such as the things mentioned above (< <b>Tab</b> >, <b>CCL Search</b> , <b>DEfine</b> and <b>Alice</b> )
lower case	Used for terms and variables where variables are also italic
Upper Case	Used for proper names, such as the database <b>Alice</b>
<b>Courier font</b>	Examples containing specific text which that is shown on the screen or is to be typed in
<>	Chevrons—used to indicate key(s) on the keyboard such as < <b>Tab</b> > or < <b>Enter</b> >
↵	Reversed 'L' arrow used after commands, meaning 'press either the keypad < <b>Enter</b> > key, or the main keyboard < <b>Return</b> > key'
< <b>Next</b> >	The < <b>Page Down</b> > or < <b>Next Page</b> > key
< <b>Prev</b> >	The < <b>Page Up</b> > or < <b>Previous Page</b> > key
< <b>Gold</b> >	The keypad < <b>PF1</b> > key (< <b>Num Lock</b> > on windows keyboards)*
< <b>Leave</b> >	The < <b>PF3</b> > key (keypad < <b>x</b> > on windows keyboards)*
" "	messages provided by TRIP

### Note:

*With the local TRIPclassic interface for Windows®, the top row keys of the keypad, <**PF1**> through <**PF4**>, are replaced by the main keyboard function keys, <**F1**> through <**F4**>.*



# CCL Command reference – Part 1:

## The interfaces

### Introduction

This reference is intended to deal with the *Tieto TRIP* (henceforth just, *TRIP*) implementation of the ISO 8777:1993 Common Command Language (CCL) and is divided into two parts.

### CCL Command reference - Part one: The interfaces

The first part of this guide deals with the structure of a CCL order and the two major CCL interfaces for TRIP, the first of these being the newer Windows based interface, *TRIPmanager* and the second, the original terminal based interface, *TRIPclassic*.

*Note:*

*CCL orders are also accessible to the various application programming interfaces (APIs); consult the relevant user guide for each API.*

### CCL Command reference - Part two: The commands

The second part of this reference, the actual order reference itself, is arranged alphabetically by CCL order, each having its own section containing a description of the order, its syntax and examples of usage.

*Notes:*

*In this reference, complex CCL orders are divided into subsections. Each order's reference starts with a header section containing the complete syntax and a brief listing of the different usages available.*

*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'.*

*Individual CCL Orders and Modifiers are represented in this reference using capitalisation to indicate the minimum acceptable abbreviation for a given item. For example, the minimum acceptable abbreviation for the CCL **DEFINE** order is the first three letters, **DEF**, hence the order is written in this document as **DEFine**. Similarly, the minimum acceptable abbreviation for the CCL **DISPLAY** order is **D** hence, in this document, the order is written as **Display**.*

*The order of precedence for CCL abbreviations is shortest first, E.g. **F** will be interpreted as a CCL **Find** order and not a **FUZZ** order.*

## Structure of a CCL Order

### Overview

A CCL *order* is made up of *constructs*. An order always begins with a *command*, which may be followed by a number of *modifiers* and/or *qualifiers*. For example, in this order:

```
Show SORT=person DESCending
```

**Show** is the command, **SORT** is the command modifier, **person** is a field and **DESCending** is a qualifier.

In each order's syntax summary, square brackets ([ ]) indicate an optional construct, braces [{ }] enclose option lists, a vertical bar [|] separates *exclusive* alternatives, and the ellipsis [...] designates a repeating construct.

For example, in the following excerpt:

```
Show [SORT=field [DESCending]]
```

either the **DESCending** or the entire [**SORT** ...] construct may be omitted.

In this example:

```
Show [USer [R={username|ALL}]]
```

either (but not both) of the constructs enclosed in braces can be used in a CCL order, and the entire **R=** construct is optional.

In the next statement:

```
Show Format=field[,field[,...]]
```

any number of *field names* in any combination may be used in any one format request.

### Maximum number of characters in a CCL Order

In TRIPclassic the considered maximum safe number of characters for a CCL is 350. This is due to *TRIPclassic* using older API functions.

*Note:*

*In actual fact, the theoretical maximum in TRIPclassic is 400 characters, but this limit may vary slightly due to the internal structure of the older C API.*

When using *TRIPmanager*, there is no limit to the number of characters permitted in a CCL order. This is because *TRIPmanager* makes use of newer API functions, where there is no such limit.

*Notes:*

- *As in TRIPnxp, there is no limit to the length of a CCL order when using the TRIPjxp API.*
- *It is also possible to avoid the 400 character limit when using the latest versions of TRIPjtk and TRIPclient; however any new TRIP session must be started using the newer TRIPcom Session object Open method, or the TRIPjtk Session interface startSession method.*
- *Details on how to use the relevant methods can be found in the documentation accompanying each API.*

- The old limit of 128 parts (atoms) in a CCL order has also been removed (see below) for both *TRIPclassic* and *TRIPmanager*.

## CCL and TRIPmanager

To open a CCL query window in *TRIPmanager*, simply choose the option "**CCL Query . . .**" from the "Action" menu, or required database or thesaurus's context menu. This will produce a window similar to Figure 1:

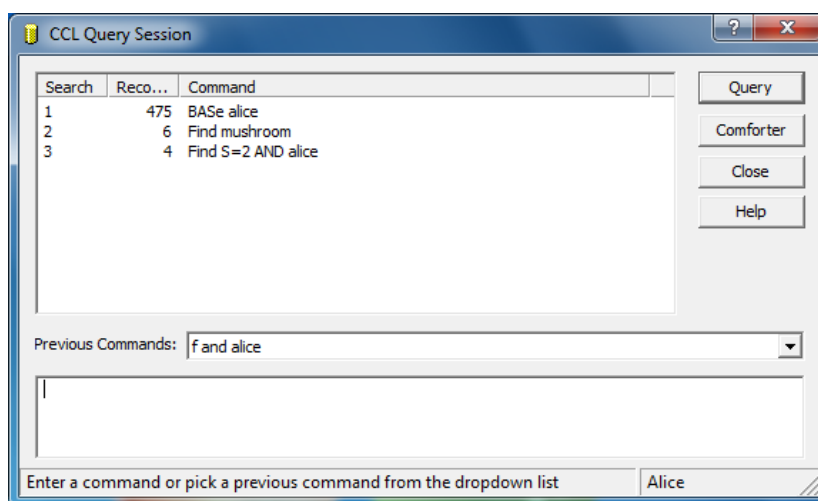


Figure 1 – The *TRIPmanager* CCL Query Window

In order to perform CCL searches, simply type the required CCL command into the text entry box at the bottom of the window and press **<Enter>**, or click the "**Query**" button at the top right of the window. All such commands, so long as they are successful, will be added to the drop-down list of "**Previous Commands**". This list equates to the command history pane in the *TRIPclassic* CCL search window and selecting a command from this list results in the command being copied to the text entry box for editing and/or (re)submission.

CCL commands that produce search sets will cause the list view box at the top of the window to update.

Commands that cause output windows to be produced, such as **Show**, **SStatus**, **FRequency**, etc. will produce any such windows accordingly.

The **Display** command produces a list of terms that match the pattern specified. For more detail, see the **Display** section in the commands reference.

**Note:**

*In TRIPmanager, the format of information such as search histories and display lists differs from that returned by TRIPclassic. TRIPmanager has column headings and the information itself is kept raw; whereas TRIPclassic, being a text based interface, displays clarification text along with the raw information, e.g. In TRIPclassic, the command history display for Figure 1 (above) would appear thus:*

```

S=1    <475>      BAsE Alice
S=2    <6>        Find mushroom
S=3    <4>        Find S=2 AND Alice

```

In order to keep the contents of this manual to within acceptable limits, all examples are given in *TRIPclassic* format; except where special attention needs to be drawn to differences in *TRIPmanager* handling or output.

## CCL and TRIPclassic

### Opening a TRIPclassic CCL Command Window

To open a CCL command windows, simply log into TRIPclassic and press the down arrow <↓> key, followed by the <Return> key. You will immediately see a TRIPclassic CCL command window as shown below in Figure 2:

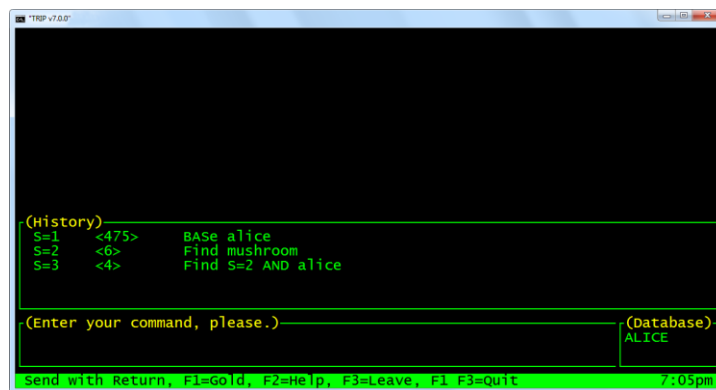


Figure 2 – The TRIPclassic CCL Command window

As can be seen above, the name of the currently open database is in the “(Database)” box to the bottom right of the window, CCL commands can be entered in the “(Enter your command, please)” box, the search history is in the box labelled “(History)” and, not shown here, search results will appear in the top part of the window.

**Note:**

*For further instructions on how to navigate the TRIPclassic menu system and on using the CCL command window shortcut key combinations, see the TRIPclassic User Guide.*

## CCL and the TRIPclassic Numeric Keypad

### Note

*The TRIPclassic keypad keys do not work with the TRIPmanager CCL query window; however, buttons are provided to substitute for any lack of keypad navigation facilities.*

The TRIPclassic CCL numeric keypad functions are diagrammed below for easy reference. Certain keys are marked with two functions: the upper one is the default, and the lower is activated when pressed immediately after the <Gold> key. These keys are defined by TRIPclassic, and may have different functions if you are using CCL through another interface, or have redefined them.

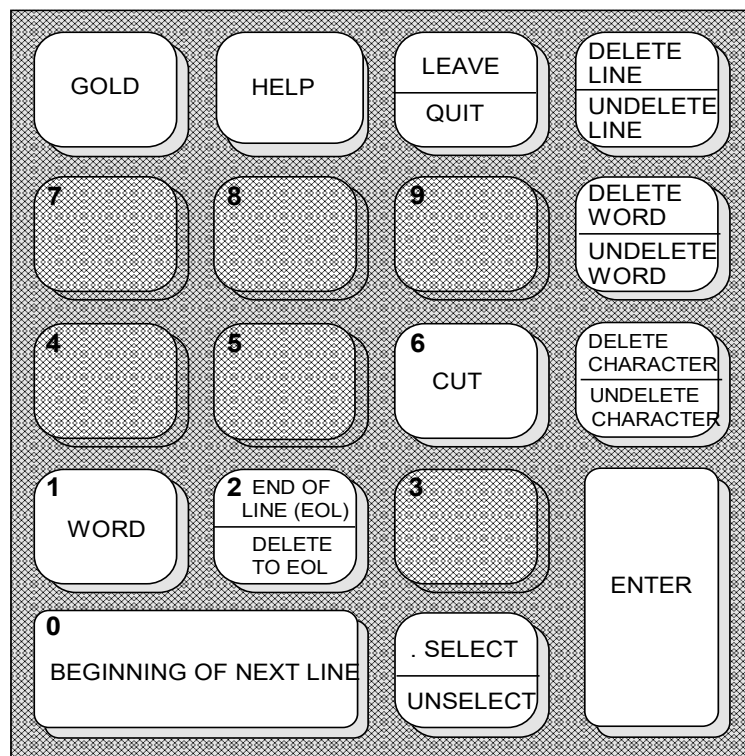


Figure 3 – TRIPclassic Numeric Keypad

### Note:

*The above keypad is for a true VT terminal keyboard, for UNIX with a standard keyboard keypad when using a terminal window and also on a standard Windows keyboard keypad when using TRIPclassic for Windows. For terminal emulator key mappings, consult your terminal emulator manual.*

## CCL Command Reference - Part 2:

### The commands

#### Back

##### Description

This order moves the cursor backward one screen, or the specified number of lines, in the displays provided by **Show**, **Display**, **Find?**, **Status**, **TRace**, **FRequency**, **MEasure** or **List** commands. If it is used after a **Show FOCUS** order, it focuses on the hit prior to the one currently shown.

##### Syntax

Back [*n*]

where *n* is a positive or negative integer representing the number of lines to move.

If Show is used with a page-oriented output format (that is, one containing form feeds, or header or trailer boxes), the 'number of lines' argument *n* is not allowed. If you attempt to use *n*, TRIP will provide a warning message and will ignore the argument.

##### Examples

*Example 1:*

Back

scrolls up (backward) one screen.

*Example 2:*

Back 5

moves back five lines of output.

*Example 3:*

Back -3

moves *down* (forward) three lines.

## BASe

### Description

A **BASe** order opens one or more databases for searching. A maximum of 250 different physical databases can be opened during a TRIP session.

When two or more databases must be opened simultaneously, they must be grouped together under another database name called a *cluster*. **Show** and **Print** orders for search results containing records from more than one database will present the records from each database in turn, in the order in which they were opened.

**BASe** may be used to open a pre-defined database or database grouping (cluster), or a database or group defined at run time. When used without modifiers or qualifiers, **BASe** displays a list of all of the databases accessible to the requesting user.

#### Notes:

*The maximum number of simultaneously open databases is 250, however in opening such a large number of databases, certain operating system limits might need to be adjusted accordingly. E.g. Open files limit.*

*This feature can be used on unmodified applications so long as the total length of a BASe command to open such a cluster does not exceed the old API 400 character limit mentioned previously on page 10.*

*The design of static database clusters is still limited to 30 database "names", however a workaround is to create database clusters containing other database clusters.*

#### Warning:

*Whilst using the workaround above, it is theoretically possible to exceed the maximum limit on simultaneously opened databases (250), however such a cluster would immediately be unusable and care should be taken to avoid exceeding this limit.*

### Syntax

To open a single database:

```
BASe [database]
```

To open a database cluster:

```
BASe [cluster[={cluster|database}[,{cluster|database}[,...]]]]
```

where *cluster* represents several single or clustered databases to be opened together, and *database* specifies a single physical database to be opened.

### Examples

Example 1:

```
BASe
```

gives an alphabetical listing of all databases to which the user has access.

Example 2:

```
S=1 <475> BASe Alice
```

opens the database called **Alice** for searching.

*Example 3:*

```
S=2      <499> BAsE group=alice,carroll
```

opens the databases **Alice** and **Carroll** as a run-time grouping in the order specified, and names this temporary cluster **Group**.

*Example 4:*

When a number of databases are open at the same time, a search order can be directed to only one of the open databases by giving the name of the open database and the field, field type or view name in which to search, separated by a period [.].

```
S=3      <574> BAsE all_demo=alice,corr
```

```
S=4      <16> Find alice.person=cheshire cat
```

These two orders name the run-time cluster called **All-demo**, which opens the databases **Alice** and **Corr** simultaneously, and search the field called **person** in database **Alice** for 'cheshire cat'.

```
S=5      <26> Find corr.TExt=new
```

This order searches all fields of type **Text** in the **Corr** database for 'new'.



## CAL

### Description

This command activates a user-written subroutine or ASE (Application Software Exit) in TRIP's CCL Command window.

### Syntax

```
CAL routine [argument[argument[...]]]
```

where *routine* is the name of the ASE to be called, and *argument* is an argument string. Everything following the routine name will be passed to the subroutine in its entirety as a single string parameter.



## CLOSE

### Description

A **CLOSE** order closes one or more databases for searching.

**CLOSE** may be used to close a pre-defined database or database grouping (cluster), or a database or group defined at run time.

Closing a database will affect the searches made in the session by nullifying the resulting hits in that particular database. The search set itself will stay in the history list.

By closing databases it is possible to exceed the limit of 250 open databases during one session.

### Syntax

*To close a single database:*

```
CLOSE database
```

*To close a database cluster:*

```
CLOSE [cluster][,][database,...]
```

where *cluster* represents several single or clustered databases to be closed together, and *database* specifies a single physical database to be closed.

### Examples

*Example 1:*

```
CLOSE alice,carroll
```

closes the databases called **Alice** and **Carroll** for searching.

*Example 2:*

```
CLOSE group
```

closes the database cluster called **Group** for searching.

## Continue

### Description

This command can be used:

after an **Expand** order during focused display and with no modifiers, to move to the next focus point

after an **Expand** order during focused display with the **Show** modifier, to return to the current focus point

with the **Edit** modifier, to return to an editing session that has previously been established using the **Edit** command, and subsequently terminated without saving.

**Continue** may also act as a modifier to **Define**, but only in procedures and macros. Refer to the section in this manual entitled 'Define Continue' for more information.

### Syntax

```
Continue [{Edit|Show}]
```

### Examples

*Example 1:*

```
S=1      <99>  BASe corr
S=2      <15>  Find TExt=product
Show FOCUS
Expand
Continue
```

opens demonstration database **Corr**, locates all records containing 'product' in a **TExt** field and displays them in **FOCUS**. The **Expand** order shows the entire focused record. The **Continue** order returns the output to the focus point immediately following the point which was expanded.

*Example 2:*

Continuing with the previous example, record two of search result two should be the active record, and shown in **FOcus**. If you then use this command:

```
EDit
```

to edit the **country** field of search set record number two (database record thirty-six) and press **<Leave>** to exit and 'Y' to retain the record lock, you will be returned to the **Show** window containing search record two.

If you decide to complete editing on the spot, use

```
Continue EDit
```

The record will be shown loaded in a data entry form as with **EDit**, with the cursor positioned in the last field edited (in this case, **country**).

*Example 3:*

```
S=3      <99>  BAsE corr
S=4      <15>  Find TExt=product
Show FOcus
Expand
Continue Show
```

which will return to the first focus point, rather than showing the second focus point, as was the case in *Example 1*.

## DEfine

### Description

**DEfine** orders are used to set the defaults for the current TRIP session. Although TRIP confirms a **DEfine** order with a message, its effect usually becomes visible when the next order is given rather than producing immediate results in itself.

If default settings other than the usual defaults are required for more TRIP sessions than not, these can be written into a start-up procedure which can then be run automatically every time a TRIP session is initiated. Activating this procedure will redefine the defaults for that user only.

A **DEfine** will remain in effect until a new **DEfine** order of the same type is given or the TRIP session is ended. For example, the order **DEfine NO Highlight** will remain active until another **DEfine Highlight** order is given or the user leaves TRIP.

Refer to Chapter Ten of the TRIPclassic User Guide for more information on defining a start procedure in your user profile and Chapter Eleven of the same manual for assistance on creating a procedure.

### DEfine – Section Index

The structure of **DEfine** orders differs slightly depending on the modifier used. Each of the modifiers listed below is covered individually in alphabetical order:

Table 1 – DEfine orders

Command	Modifier
DEfine	A Simple Order
DEfine	, (Comma)
DEfine	ABout
DEfine	AND
Define	CENTury MINimum
DEfine	COMmand KEY
DEfine	Continue
DEfine	COPy
DEfine	CUt
DEfine	Delete
DEfine	Display – See <i>DEfine</i> MAXimum/MINimum
DEfine	EForm
DEfine	Find– See <i>DEfine</i> MAXimum/MINimum
DEfine	FOcus
DEfine	Format
DEfine	FUZz
DEfine	HIghlight
DEfine	HOLD
DEfine	KEY
DEfine	KVP
DEfine	LPcode
DEfine	MAP – Also see <i>DEfine</i> MAXimum/MINimum

Command	Modifier
DEfine	MASK
DEfine	MAXimum/MINimum
DEfine	MERGE
DEfine	Page
DEfine	PCode
DEfine	Print – See <i>DEfine</i> MAXimum/MINimum
DEfine	PRINTER
DEfine	REVerse
DEfine	SAve BASE
DEfine	SCope
DEfine	SCope SDI
DEfine	SORT – See <i>DEfine</i> MAXimum/MINimum
DEfine	SPace
DEfine	STop WOrd
DEfine	THESaurus
DEfine	TIMEForm
DEfine	<b>Fel! Hittar inte referenskälla.</b>
DEfine	Vlew
DEfine	WEIght

## Define – A Simple Order

### Description

This simplest form of **DEfine** gives a list of the current default settings (e.g. for **Show** and **Find**) as well as the current function key assignments.

Short Form

DE?

### Syntax

DEfine?



## DEfine , (Comma)

### Description

The comma character can be defined as synonymous with the Boolean **OR/XOR** operators, any of the Boolean **AND** operators or the proximity operator for use with **Find** orders. Comma can also be defined as a word mask.

The system default is for the comma to be defined as itself (a comma, [,]), and as such is ignored in **Text** or **Phrase** field searches.

For a list of all possible Boolean operators, refer to the **Find** command sections presented later in this guide.

### Syntax

```
DEfine , [= [{OR|AND.x | (y..z) }]]
```

where **OR** and **AND.x** are Boolean operators, and y and z are values indicating the allowable number of terms that may appear between two search target terms. If no qualifier is specified, the comma is reset to the system default value.

### Examples

#### Example 1:

```
DEfine , =OR
```

defines the comma character to represent the operator **OR**. When the next two **Find** orders follow a **DEfine ,=OR** statement, they become equivalent:

```
S=1      <475> BAsE Alice
S=2      <21> Find dormouse,alice,queen AND cat
S=3      <21> Find (dormouse OR Alice OR queen) AND cat
```

#### Example 2:

```
DEfine , =AND.S
```

defines the comma character to represent the operator **AND.S**. When the next two **Find** orders follow the above **DEfine ,=AND** statement, they become equivalent:

```
S=4      <99> BAsE corr
S=5      <10> Find 3rip,commercial OR free
S=6      <10> Find 3rip AND.S commercial OR free
```



**Example 3:**

```
DEfine ,=AND.P
```

defines the comma character to represent the operator **AND.P**. When the next two **Find** orders follow a **DEfine ,=AND.P** statement, they become equivalent:

```
S=7      <475> BAsE Alice
S=8      <14>  Find queen,alice AND.S red
S=9      <14>  Find queen AND.P Alice AND.S red
```

**Example 4:**

```
S=10     <475> BAsE Alice
DEfine ,=(-1..0)
S=11     <34>  Find rabbit,white
```

This **DEfine** order causes the comma character to represent the proximity operator. The search order then finds terms in either order, and in direct proximity.

The **Find** order would thus locate the phrase ‘...suddenly a *white rabbit*...’, but not the segment ‘...*rabbit* of *white* fur...’

**Example 5:**

Both of the following orders:

```
DEfine ,
DEfine ,=
```

restore the TRIP system default, ‘no **DEfine**’. Except in special cases where this is defined in the database design, the comma character is ignored when searching in TRIP’s **Text** and **Phrase** fields.

Commas will, however, produce valid search results with **Number**, **Integer**, **Time** or **Date** fields, where the comma is treated as **OR**. This is shown in the series below:

```
S=12 <475>  BAsE Alice
S=13 <81>   Find chaptnr=1,2
```

which locates eighty-one records where the value of **chaptnr** is either ‘1’ or ‘2’. A **DEfine** order has no effect when searching numeric field types.

## DEfine ABout

### Uses:

- a) Defines the precision in matching to be used for non-Boolean search.
- b) Toggles highlighting of terms from a non-Boolean search. (Default is off.)

### Syntax:

```
DEfine ABout=precision [, [NO] HIghlight]
```

where *precision* defines an “acceptable” level of match between query and documents found. It is given as a percentage value between 1 and 100 and the default value is 60.

*Notes on non-Boolean rank calculation:*

*Weighting co-occurrence of terms, i.e. if more than one term from the query is present in the document, gives higher rank so as to bias multi-term hits.*

*The definable search precision percentage (DEfine ABout=n) is now Based on highest ranked record.*

*Define About=n, means that records with less relevance are discarded.*

*Records with a weight of n% of the highest rank will be kept.*

### Examples

The following examples use a news articles database called **tipster**.

#### Example 1:

```
S=1      <10000>    BAsE tipster
S=2      <27>    Find about (peanut butter)
S=3      <49>    Find about (r=8163)
DEfine About=80
S=4      <3>    Find about (r=8163)
Define About=60
S=5      <68>    Find about (r=8163)
```

## DEfine AND

### Description

A record may consist of a head record and any number of part or sub-records. The *head record* contains one or more *head fields*, and each *part record* one or more *part fields*. A *record component* comprises a head record or one part record, and a *record entity* is the union of one part record and its head record.

Several **AND** operators that can make searching head and part record databases more sensitive may be fixed for the duration of a search session using **DEfine**, as outlined below:

Table 2 – Operators for the order DEfine AND

Operator	Search Function
AND.R	AND within a record (head plus all parts)
AND.E	AND within an entity (head plus one part)
AND.C	AND within a component (head <i>or</i> part)

By system default, **AND** is equivalent to **AND.E**.

When **DEfine** is used with **AND**, **NOT** is automatically assigned the corresponding interpretation. For example, if **AND** is defined as **AND.C**, then **NOT** is redefined as **NOT.C** as well.

Refer to Chapter Eight, 'Head and Part Records' in the *TRIPclassic User Guide* for further information regarding head and part records.

### Syntax

```
DEfine AND=AND.x
```

where x is the letter suffix of the desired head/part **AND** operator.

### Examples

The following examples use the demonstration database **Carroll**, and are meant to be worked consecutively.

*Example 1:*

```
S=1      <24>  BASE carroll
```

```
DEfine AND=AND.R
```

```
S=2      <1>   Find white bread AND brown bread
```

finds all records in demonstration database **Carroll** where the terms 'white bread' and 'brown bread' are mentioned anywhere within the same record.

*Example 2:*

```
DEfine AND=AND.E
```

```
S=3      <1>   Find looking glass AND insect
```

finds all records in **Carroll** in which the terms 'looking glass' and 'insect' are found in any head-plus-part pairing or entity.

*Example 3:*

```
DEfine AND=AND.C
```

```
S=4      <2>   Find rabbit AND kid gloves
```

finds 'rabbit' and 'kid gloves' in the same head or part record of **Carroll**.



## DEFine CENTury MINimum

### Description

TRIP *always* saves four digits for the year in dates, even if the user only supplies 2 digits. The system variable that holds the value (the century) to be added to a 2-digit year can be set in two different ways. The first method sets the value system wide (described in the TRIP System Manager Guide); the second method changes it locally for the current TRIP-session only.

### Syntax

To set the choice of centuries to be added to a 2 digit year when storing the value in a TRIP database.

```
DEFine CENTury MINimum = n
```

The parameter *n* allows one of two centuries to be supplied depending on the value of the parameter and the two digits given by the user: if the parameter is  $X*100 + Y$  and the given digits form the number *Z*, then the century will be *X* for  $Z \geq Y$  and  $X+1$  for  $Z < Y$ . The parameter can also be expressed relative to the current year by a negative two digit value. This value is then added to the current date to form a four digit century start as described above.

### Examples

*Example 1:*

```
DEFine CENTury MINimum = 1900
```

means that TRIP behaves just as it has done in earlier versions – i.e. for any date for which the user has only supplied two digits for the year, 1900 will be added to the year.

*Example 2:*

```
DEFine CENTury MINimum = 1978
```

means that if the two digits the user has supplied for the year are 78 or greater then 1900 will be added to the year, otherwise 2000 will be added.

*Example 3:*

```
DEFine CENTury MINimum = -38
```

Rather than defining a fixed breakpoint, such as 78 in the example above, a floating breakpoint may be defined by use of a hyphen. In this example the breakpoint will at any given time be set to the current year minus 38 i.e. in 1998 the line will be interpreted in the same way as:

```
DEFine CENTury MINimum = 1960
```

and in 1999 it will be interpreted as:

```
DEFine CENTury MINimum = 1961
```

## DEfine COMmand KEY (TRIPclassic only)

### Description

The keys F7 through F10 (and F14 through F20 on VT200 keyboards) may be defined as *soft function keys*, where pressing a predefined key causes TRIP to call an ASE (Application Software Exit) or execute a CCL order.

As when typing such commands in the CCL window, each will be *echoed* or displayed onscreen following its execution unless an underscore [ \_ ] is positioned immediately after the equality symbol [=].

Use **DEfine?** to review current soft function key definitions.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

*To assign a CCL order to a function key:*

```
DEfine COMmand KEY key=[_]CCLorder
```

*To execute an ASE using a function key:*

```
DEfine COMmand KEY key=\ase  
[argument[argument[...]]]
```

where *key* is the alphanumeric name of the key to be defined, *CCLorder* is a complete CCL order, *ase* (always preceded by a backslash [\]) is the name of an ASE to be called and *argument* is the value of an argument to be passed to the ASE routine.

### Examples

The following examples use the demonstration database **Corr**, and are meant to be worked consecutively.

*Example 1:*

```
S=1      <99>  BASe corr  
DEfine COMmand KEY F9=Show SORT=rname
```

defines the F9 key to sort the current search result by the field called **rname** in database **Corr** and display it in the **Show** window.

*Example 2:*

```
DEfine COMmand KEY F9=_Show SORT=rname
```

defines the F9 key to sort the current search result by the **Corr** field **rname** and display it in the **Show** window whenever F9 is pressed, but not to display the command in the command window.

*Example 3:*

```
DEfine COMmand KEY F10=Find product AND (tdbs or trip or  
3rip)
```

carries out the **Corr**-based **Find** order included in the **DE COM KEY** statement whenever the F10 key is pressed.

## DEfine Continue

### Description

Normally, any error produced in a procedure or macro causes that procedure or macro to terminate. **DEfine Continue** allows procedures and macros to continue execution regardless of the number or variety of errors incurred.

### Syntax

```
DEfine [NO] Continue
```

### Examples

#### *Example 1:*

```
DEfine Continue
```

prevents a procedure or macro from terminating in the event of an error.

#### *Example 2:*

```
DEfine NO Continue
```

resets **Continue** to normal, where processing terminates on error.

## DEfine COpy

### Description

This order specifies a target database to receive records copied from the same or another database.

If the source of the **COpy** operation is a different database from that defined, only those fields whose names match will be copied. If any fields have incompatible types between the two databases, the copy operation will not be allowed.

### Syntax

```
DEfine COpy=database
```

where *database* is the database to receive copied records.

### Examples

#### Example 1:

```
DEfine COpy=corr
S=1      <99>  BAsE corr
S=2      <15>  Find product
S=3      <3>   Find S=2 AND trip
EDit COpy S=3
```

This series makes **Corr** the copy destination database, then performs a number of searches and loads the records found by search result number three into a data entry form for editing and copying into destination database **Corr**.

The original records found by search result number three are unaffected by the editing. The resulting records are instead added to the end of the database as new records.



## DEfine CUt

### Description

This order specifies a text file that will store one or more consecutive lines copied from the display of a **Show** window.

Once the file has been defined, you can select, cut and store text in that file using the following steps:

1. If the cursor is not already positioned somewhere in the **Show** window, move it there using the **<Select>** key
2. position the cursor at the first line of the region to be copied using the **<↑ Arrow>**, **<↓ Arrow>**, **<Next>** and **<Prev>** keys, and press **<Select>**
3. place the cursor at the end of the desired section and press **<CUt>**
4. the selected area is written to the text file specified, and the cursor returns to the command window.

### Syntax

```
DEfine CUt=file
```

where *file* is the name of the file to which the selected portion will be copied.

### Examples

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <1>   Find rath
DEfine CUt=misc.txt
Show
```

**<Select>** the sentences “Well, a ‘RATH’ is a sort of green pig: but ‘MOME’ I’m not certain about. I think it’s short for ‘from home’—meaning that they’d lost their way, you know.”

Press **<CUt>**, exit to a shell and read or edit the file ‘misc.txt’ as desired. If ‘misc.txt’ is an existing file, then the cut pieces will be appended to it.

## Define Delete

### Description

In order to avoid deleting the complete content of a database by mistake the global record delete can now have a maximum value defined.

### Syntax

```
DEFine Delete Max = [nnn | NO LIMIT]
```

where *nnn* is the maximum number of record that may be deleted using a global update delete command. Any attempt to delete more than the defined maximum will result in an error similar to the following:

```
Too many records (475) for DELETE, current limit is 100.
```

### Notes:

*Unless the default value for *nnn* is set using a startup procedure or defined in the *tdbs.conf* file using the environment variable *TDBS\_DELETE\_MAX=nnn* it will only apply to the current session.*

*The default value for *nnn* is 100*

*See the Delete – Global Update section for more details on using this command.*

## DEfine Display

### Descriptions

1. Enables/disables merge across databases of terms in frequency restricted term lists.
2. Define DISPLAY lists to use either normalized or original forms of the terms.
3. Define the displaying of field content from up to 10 tupled fields

### Syntax (1):

```
DEfine Display FReq=[NO] MERGe
```

*Notes on syntax (1):*

*Default is "MERGe".*

*This value can also be set in the `tdbs.conf` file using either*

*`TDBS_DISPLAY_FREQ=M` (merge)*

*or*

*`TDBS_DISPLAY_FREQ=N` (no merge).*

*Using "merge" with very large database clusters might cause long execution time and possibly also heavy memory usage.*

### Syntax (2):

```
DEfine Display [NO] ORIG
```

### Syntax (3):

```
Define Display FIELD=f2,f3,...
```

*Note on syntax (3):*

*For more information on **Display FIELD**, refer to the CCL Display order , **Enhanced Output** section, starting on page 99 of this reference.*

*General note:*

*For more information on `DEfine Display`, refer to the section entitled: **DEfine MAXimum/MINimum***

## DEfine EForm (TRIPclasic only)

### Description

Although any number of data entry forms may have been created for any one database, each database usually has a default entry form defined for use during those times when no specific entry form has been requested. Use the order **DEfine EForm** to change the predetermined default entry form.

*Note:*

*This command is valid in TRIPclasic only.*

### Syntax

```
DEfine EForm=entryform
```

where *entryform* is the name of the data entry form to define as the default.

### Examples

*Example 1:*

```
S=1      <24>  BAsE carroll  
DEfine EForm=1
```

changes the default entry form for the database **Carroll** to '1' for the current user session only. The **DEfine** order does not affect the database design.

*Example 2:*

```
DEfine?
```

displays, among other things, the most recent default entry form definition.

## DEfine Find (FUZz)

### Description

The **DEfine Find FUZz** command permits the simple **Find** command to perform a 'web style' search by default, behaving as if it were the **Fuzz()** command.

### Syntax

```
DEfine Find [=] [NO] FUZz
```

#### Notes:

- When the command "**DEfine Find FUZz**" is issued, the status message "*FIND command enabled with web search features.*" will confirm correct execution.
- When the command "**DEfine Find NO FUZz**" is issued, the status message "*FIND command enabled without web search features.*" will confirm correct execution.
- For more information regarding this command, please refer to the section in this manual entitled '**FUZZ**'.

## DEfine Find (Max/Min)

### Description

For more information on this CCL order, refer to the section entitled:  
Define – **MAXimum/MINimum**



## DEfine FOCUS

### Description

**FOCUS** is a modifier for **Show** and **Print** orders, and is used to present only hit terms and their immediate surroundings on output, i.e. their paragraphs or subfields.

The symbol **NO** cancels focusing. Use the **DEfine FOCUS** order to change the **NO FOCUS** system default to **FOCUS**, and **DEfine NO FOCUS** to return to unfocused output.

*Note:*

***FOCUS** works similarly to **HIGHLIGHT**, in that they are both dependent on the currently defined output format. If all occurrences of the term to be focused exist in fields not included in the output format, the output result will be empty.*

For further examples of how the modifier **FOCUS** and the command **Expand** work, refer to the section entitled 'Focus - Showing Only the Hits in Records' of Chapter Three, 'Tutorial Three' in the *TRIPclassic User Guide*.

### Syntax

```
DEfine [NO] FOCUS
```

### Examples

Consider these orders as having been given consecutively.

*Example 1:*

```
S=1      <475> BAsE Alice
DEfine FOCUS
S=2      <56> Find rabbit
Show
```

shows all subsequent search results in **FOCUS**, in which only the paragraphs or subfields containing the hit terms are displayed.

*Example 2:*

```
Expand
```

shows the current focused record (if any) in its entirety. You will be unable to view any other records until you give a new **Show** order or use the **Continue** command.

*Example 3:*

```
DEfine NO FOCUS
```

cancels the focusing effect, and causes all subsequent search results to be shown without **FOCUS**.

## DEfine Format

### Description

Although any number of output formats may be available for a particular database, each may have a default output format defined for use when no output format is specified. Use the order **DEfine Format** to change the default output format. This format will remain the default for the session until another **DEfine Format** order is given or the search session ends.

Pre-defined or 'named' formats are created by database administrators, and those available for any database can be seen in the **STatus** list. The format 'Dump' is a pre-defined format supplied with TRIP, and does not appear in the **STatus** list..

Run-time formats can be defined directly, and are essentially a list of the names and/or types of the fields to be output, separated by commas. The available field names and types for any database can be seen in the **STatus** list.

If no format is specified in a **Show** order after a **DEfine Format** statement, the one given in the **DEfine** request is used. Both pre-defined and run-time output formats can be made the default in this way.

A **Format** modifier may be included in a **Show** order to temporarily present the results in a format other than the default. Refer to the section entitled 'Show Format' in this guide for further information.

Although the two types of field qualifier, field name and field type, can be mixed in the same **Show** order, the two types of format, pre-defined and run-time, cannot.

### Syntax

*For pre-defined formats:*

```
DEfine Format=format
```

where *format* is the named format to be used.

*For run-time formats:*

```
DEfine Format={field|fieldtype|view}  
[, {field|fieldtype|view}[,...]]
```

where *field/fieldtype* may be either a field name, **TStamp** or field of type **Text**, **PHrase**, **NUmber**, **DAte** or **Time**.

### Examples

These examples have all been created using the **Alice** demonstration database provided with TRIP:

*Example 1:*

```
S=1      <475> BASe Alice  
DEfine Format=chaptnr,person  
S=2      <24> Find unicorn  
Show
```

defines the default output format to write only the fields **chaptnr** and **person**, where these are not empty. The content is preceded by the field name, and the output order of



the fields is the same as in the **DEfine** order; i.e. in this case **chaptnr** appears before **person**.

*Example 2:*

```
DEfine Format=chaptnr, PHrase  
Show
```

*defines the default output format to print only **chaptnr** and the non-empty **PHrase** fields. The output order of the fields is the same as in the **DEfine** order, with the output order of the **PHrase** fields as they are on the database.*

*Example 3:*

```
DEfine Format=2  
Show
```

defines the default output format to be the predefined format named '2'.

*Example 4:*

```
DEfine Format=dump  
Show
```

defines the default output format to be the TRIP default format, which shows the contents of *all* the fields on the database where these are not empty. The content is preceded by the field name, and the fields are output according to their type in this order: **PHrase**, **NUmber**, **INteger**, **DAte**, **TIme** and **TEExt**.

## DEfine FUZZ

### Description

*Note:*

*The command **DEfine FUZZ** and the **FUZZ()** modifier used in **Find FUZZ** and **Display FUZZ()** should not be confused with the CCL '**FUZZ**' command, which performs a different role within TRIP.*

Fuzzy searching implies searching for similarity, a capability that can prove quite useful when looking for differently spelled or frequently misspelled terms.

The **DEfine FUZZ** order allows the four parameters associated with CCL's **FUZZ()** modifier; i.e. **Find FUZZ & Display FUZZ()**; to be defined at any time during a search session.

### Parameters

1. 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

2. The second parameter is depreciated and has no effect in normal fuzz searching; however, when using **Define FUZZ**, an integer value of 1 must be entered if any of the following parameters are also to be used.
3. The third parameter sets the number of top ranked search candidates to include in **Find FUZZ()**.

Default Value: 2

*Note:*

*See the **Display FUZZ()** command for details on how to obtain a list of search candidates used in a **Find FUZZ** search.*

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

Default: 1

*Note:*

*The default is also the recommended search algorithm.*

Refer to the section **Define FUZZ** in this manual for more information on redefining the default values of fuzzy parameters, and **Display FUZZ()** for help with listing fuzzy terms.

### Syntax

```
DEfine FUZZ=param1[,param2,param3,param4]
```

## Examples

### Example 1:

```
DEfine FUZZ=50,1,4,2
```

After this **DEfine** order, *all* subsequent **Find FUZZ** and **Display FUZZ()** orders will use the 'sliding mask' algorithm to return only terms which have a fifty percent similarity to any query term. For a **Find** command *only*, the four top ranked search candidates will be used when searching.



## DEfine Highlight

### Description

Hit terms within output results may be highlighted to help them stand out from their surroundings, which is useful when there are a relatively small number of hit terms within a record.

**DEfine** orders affect output according to the type of output (screen, printer or file) that is required. Depending on the modifier used, if given before a **Find** order using one of the logical operators, then output to screen using **Show**, the **DEfine** command will have one of the effects listed below:

Table 3 – Modifiers for the order DEfine Highlight

DEfine Command	Operator	Terms Highlighted
DEfine Hlghlight	AND	only the rightmost search term
	OR, XOR	all search terms
	NOT	only the leftmost search term
DEfine Hlghlight=ALL	AND, OR, XOR	all terms
DEfine Hlghlight=MIN	All operators	Highlight all hits except those from searches for non-empty fields
DEfine No Highlight	AND, OR, XOR, NOT	no search terms

If given before a **Find** order, then output to a printer using **Print**, the **DEfine** orders will have no effect. If given before a **Find** and a **Print Highlight** order, they will have the same effect as with **Show**. see the section on **Print Highlight** in this manual for more information.

The default is **Hlghlight=ALL**, which causes *all* hit terms to be highlighted in **Show** orders. Of necessity, there can be no highlighting effect if all hit terms occur in fields not shown by the prevailing output format.

The symbol **NO** cancels highlighting for the balance of the search session, until another **DEfine** order is given. Session highlighting may be re-established using **DEfine Hlghlight**.

A **Hlghlight/NO Hlghlight** modifier may be included in a **Show** order to temporarily override the default for the highlighting of hit terms. Refer to the section entitled 'Show Highlight' in this guide for further information.

### Syntax

```
DEfine {[NO] Hlghlight|Hlghlight=ALL|MIN}
```

### Examples

The following examples use the demonstration database **Alice**, and should be worked consecutively.

*Example 1:*

```
DEfine NO Hlghlight
```

defines **NO Highlight** to be the default, and shows the results of subsequent search results hit terms unhighlighted.

*Example 2:*

```
S=1      <475> BAsE Alice
DEfine HIghlight
S=2      <1>   Find rabbit AND watch AND pocket
Show
```

causes the right-most hit term 'pocket' to be shown highlighted in this **ANDed** search request.

*Example 3:*

```
DEfine Highlight=ALL
S=3      <1>   Find rabbit AND watch AND pocket
Show
```

shows and highlights *all* hit terms found by each search term in combination search requests using **AND**.

## DEfine HOLD

### Description

**HOLD** is used to define the default for **Print** requests, and is the initial session default. This means that **Print** orders are held by TRIP until the end of the TRIP session, and then submitted as one batch job to the print queue. **NO HOLD** means that as each **Print** order is given, it is submitted immediately to the print queue.

There are other modifiers which affect the immediacy of the execution of **Print** orders. These are covered in greater detail in the section called 'Print–Timing' in this manual.

### Syntax

```
DEfine [NO] HOLD
```

### Examples

*Example 1:*

```
DEfine HOLD
```

executes all submitted print jobs upon leaving TRIP.

*Example 2:*

```
DEfine NO HOLD
```

carries out each print request as it is submitted.

## DEfine KEY (TRIPclassic only)

### Description

The keys F7 through F10 (and F14 through F20 on VT200 keyboards) may be defined as *soft function keys* or *macro keys*. These user-defined keys cause TRIP to insert a predefined text string in the current order, or execute an ASE when the key is pressed.

Use **DEfine?** to review current soft function key definitions.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

To write a text string to the CCL window using a function key:

```
DEfine KEY key=textstring
```

To execute an ASE using a function key:

```
DEfine KEY key=\ase [argument[argument[...]]]
```

where *key* is the alphanumeric name of the key to be defined, *textstring* is a piece of text to be inserted into the CCL order under construction and *ase* (always preceded by a backslash [\\]) is the name of an ASE to be called.

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr
```

```
DEfine KEY F9=(tdbs OR trip OR 3rip)
```

If you type a partial CCL order in the command window, such as

```
Find product AND ٠
```

and press F9, the desired string is inserted in the order. If you then press **<Return>** or **<Enter>**, the new composite order

```
S=2      <13>  Find product AND (tdbs OR trip OR 3rip)
```

is executed. The symbol [٠] indicates a single space.

## DEfine KVP

### Description

Key-value pair Find/Display takes a value from one field (key) and treats it as a field name for the values of another field (value) within the same tuple.

### Syntax

```
Define kvp=key_fld, value_fld
```

For more information on this CCL order, refer to the sections entitled: **Find** and **Display**, on pages 125 and 92 respectively of this reference.





## DEfine LPcode

*Note:*

*This command is not available for Windows.*

### Description

**LPcode** is used to select a translation table for use with **Print Local** orders.

### Syntax

```
DEfine LPcode=file
```

where *file* is the text file name containing the desired translation table. These files should reside in the directory pointed to by \$TDBS\_PRC for UNIX.

### Examples

*Example 1:*

```
DEfine LPcode=ascii_eng
```

specifies that output to the local printer is to be processed using the translation table stored in the UNIX text file \$TDBS\_PRC/ASCII\_ENG.PRC.

## DEfine MAP

### Description

An indirect search order takes the vocabulary of one numeric field, **PHrase** field, field view or phrase, from a set of records in one database (the source) and applies these as search terms to the numeric, **Text** or **PHrase** fields of another database (the target). **MAP** is used in **DEfine** orders to specify a *virtual field* to be used in indirect searching. A virtual field is a temporary or 'ghost' field created for convenience, and does not truly exist as part of any database definition.

### Syntax

```
DEfine MAP [virtualfield]=[{fieldtype|field|view}:]  
database.sourcefield [{:fieldtype|field|view}]
```

where *virtualfield* is the temporary, predefined search field, *fieldtype* may be **PHrase**, **Text**, or **Text+PHrase**, *field* is the name of a numeric, **Text** or **PHrase** field, *view* is a view across **Text** and/or **PHrase** fields and *sourcefield* must be a **PHrase** field of the database specified by *database*.

In the context of this statement, the:

- *database* is the source database,
- *sourcefield* is one of the source database's **PHrase** fields, field views or generic fieldtype **PHrase**
- *fieldtype/field/view* preceding the database are the fields of the source database to which the search term is applied,
- *fieldtype/field/view* after the sourcefield are the fields of the target database to which the extracted terms are applied,
- and *virtualfield* is the name given by the user to reference the **PHrase** field in the source database from which terms are to be extracted.

If the source and/or target *field(s)* are omitted, searching occurs by default in all **Text** and **PHrase** fields. If the *virtualfield* is omitted, the original source database field name must be used.

When exact matching of a search phrase and the target subfield is necessary, place the name of the target field within single quotation marks in the **DEfine** order.

The target database is the database currently open for searching. It may be a cluster, and it may be the same as, or include, the source database.

There is a limit to the number of source records from which search terms may be extracted. The default **MAXimum** number of 1000 can be set or changed using the order:

```
DEfine MAP MAX=n
```

where *n* is an integer with a value of 10 or greater. See the section called **DEfine MAXimum/Minimum** in this guide for more information.

Further information regarding indirect searching can be found in the 'Find Mapped Transaction Sets' section of this manual, and in Chapter Seven of the *TRIPclassic User Guide*, 'Indirect Searching'.

## Examples

### Example 1:

```
S=1      <99>  BAsE corr
S=2      <0>   DEfine MAP=corr.SNAME
S=3      <77>  Find sname(mats)
```

specifies that the field called **sname** in the demonstration database **Corr** will act both as virtual field and the field that contains the search terms for the target database(s). When no other parameters are specified, as in this **DEfine** order, source database searching as well as searching for extracted terms in the target database is done in the **Text** and **PHrase** fields.

### Example 2:

An indirect search can be used to find all of the records in database **Alice** which mention 'rabbit'. The contents of the **person** field (type **PHrase**) of each hit record will then be applied to the **Carroll** database:

```
S=1      <24>  BAsE carroll
S=2      <0>   DEfine MAP people=alice.person
S=3      <17>  Find people(rabbit)
```

This series of orders opens the **Carroll** database, maps the contents of the **Alice PHrase** field **person** to the virtual field **people**, and finds the occurrences of 'rabbit' in the **Text** and **PHrase** fields of **Alice**. It then takes the values from the **person** field of the hit records and applies these as search terms to the **Text** and **PHrase** fields of **Carroll**.

### Example 3:

```
S=1      <24>  BAsE carroll
S=2      <0>   DEfine MAP people=PHrase:alice.person:Text
S=3      <16>  Find people(rabbit)
```

maps the contents of the **PHrase** field **person** of database **Alice** to the virtual field **people**, and finds all occurrences of 'rabbit' in **Alice's PHrase** fields. It then takes the values from the **person** field of the hit records and applies these as search terms to the **Text** fields of database **Carroll**.

### Example 4:

```
S=4      <0>   DEfine MAP=sname:corr.rname:PHrase
```

Indirect searches given after this **DEfine** request will locate records in the demonstration database **Corr** that have occurrences of the search term in the **sname** field. TRIP will then search for terms extracted from **rname** in the **PHrase** fields of the target database(s).

## DEfine MASK

### Description

TRIP provides three methods of blocking unknown words or word parts while searching: *word masking*, *word truncation* and *character masking*.

*Word masking* allows TRIP to ignore one or more words that may occur between any two search terms. For example, record number 375 in the demonstration database **Alice** contains the sentence ‘ “Well, a ‘*Rath*’ is a sort of green pig: but ‘*Mome*’ I’m not certain about.” ’. The search order

```
Find rath $ $ $ $ $ $ $ mome
```

will locate this sentence by substituting one dollar symbol [\$] for each of the seven words that separates the search terms ‘mome’ and ‘rath’.

*Word truncation* ‘abbreviates’ a word by one or more characters by concealing these prefix or suffix characters from the search. For example, to find all the terms beginning with ‘ja’ in **Alice**, you could use the order

```
Find ja#
```

Substituting the pound symbol [#] for all characters after the ‘a’ causes all such characters to be accepted in a search, making this request locate terms such as ‘jar’, ‘Jack’, ‘jam’, ‘Jabberwock’ and ‘Jabberwocky’.

To find all the words ending in ‘tation’, the order

```
Find #tation
```

would ignore all characters preceding the first ‘t’ and might locate records containing ‘station’, ‘consultation’, ‘invitation’ and ‘irritation’.

*Character masking* blocks one or more characters within a word. For example, the order

```
Find co#r
```

causes all characters between the ‘o’ and the ‘r’ to be accepted, and so locates terms such as ‘collar’, ‘colour’, ‘conger’, ‘conqueror’, ‘consider’, ‘corner’, and ‘counter’.

The four system default truncating and masking characters have the initial settings shown below:

Table 4 – Default truncators and masks for the FIND order

Mask Position	Initial Value	Mask Type
1	#	any number or sequence of characters
2	:	zero or one characters
3	!	exactly one character
4	&	exactly zero words

In a **DEfine MASK** order, the system default characters are always specified in this sequence: [# : ! &]. Each mask type can be used as it is, left unchanged by substituting a space character, cancelled by replacing it in the string of mask characters by a zero or redefined as a different mask character. If the masking string is shorter than four characters, any unspecified mask characters are cancelled.

The pound symbol [#] is used both as a word mask and as a truncating and masking character. As a word mask, it represents any number of words. As a truncating and

masking character, the pound symbol signifies a character string of any length, including zero, irrespective of its position within the word.

The colon [:] is used in **Find** and **Display** orders as a masking character, denoting one or zero characters.

The exclamation point [!] depicts one character in **Find** and **Display** requests.

The ampersand [&] is used in **Find** and **Display** orders as an anchor. The word following the [&] must appear first in its subfield or sentence to fulfil the search condition.

In addition to these four, seven other characters can be used for masking: the underscore [\_], semicolon [;], asterisk [\*], 'at' symbol [@], left single quote ['], question mark [?] and slash [/]. There are also two permanent masking and truncation characters that may not be redefined: the dollar symbol [\$] and full stop or period [.]

*Note:*

*The additional seven characters given above may be used for masking only if they have not been defined as searchable characters in the database design.*

## Syntax

```
DEfine MASK [= 'maskcharacter1[maskcharacter2[maskcharacter3  
[maskcharacter4]]]']
```

where maskcharacter1 ... 4 represent the masking symbols to be redefined.

## Examples

*Example 1:*

```
DEfine MASK
```

restores all mask characters to their initial values or defaults.

*Example 2:*

```
DEfine MASK='* ; ? @'
```

defines the masking characters to be [\* ; ? @], in place of the default characters [# : ! &].

*Example 3:*

If the default mask [# : ! &] is in effect, and the command

```
DEfine MASK=' ;@ '
```

is given, then the masking character [:] is redefined as [;] and [&] is defined as [@]. Each [ ] symbol represents a space in the string, one replacing the [#] symbol and one the [!]. The [#] and [!] symbols or their current substitutes remain unchanged.

*Example 4:*

If the default mask [# : ! &] is in effect, and the command

```
DEfine MASK='0;0@ '
```

is given, the masking character [:] is redefined as [;], and the [&] as [@]. One [0] symbol replaces the [#] symbol and one the [!], therefore cancelling them.

*Example 5:*

```
DEfine MASK='# '
```

This command restores the first masking character [#] to the system default.

*Example 6:*

```
DEfine MASK='# : ! & '
```

This command restores all masking characters to the system default.

*Example 7:*

```
DEfine MASK='0 '
```

cancels all mask characters. All masking characters are now treated as non-searchable characters or spaces, thus disallowing any masking.

## DEfine MAXimum/MINimum

### Description

The **DEfine MAXimum** and **DEfine MINimum** orders restrict the number of terms that may be hit when searching or displaying, and the number of records that can be printed or sorted at any one time.

If TRIP reaches the predefined maximum or minimum values while processing an order, it stops and informs the user of the reason for the premature termination. The user can then resubmit the order using values that are consistent with the predefined maximums or minimums, or redefine **MAXimum** and **MINimum**.

**DEfine MAXimum** may be used with the **Find**, **Display** and **Print** commands and the **MAP** and **SORT** modifiers. The range of permissible **MAXimum** values is from 10 up to the machine-dependent maximum integer or "*No limit*". When setting the **MAXimum** value to "*No limit*" the actual limit will be the amount of memory currently available. The default **MAXimum** values are listed in Table 5, below.

**DEfine MINimum** is available only for **Find**, and accepts values of zero or one. The default value is one, where search orders giving zero hits are not saved as part of the search history list. On occasion, it may be useful to save these 'empty' or unsuccessful searches to a procedure for later use, perhaps when the database has changed and the empty search is more likely to produce hits. Specifying a **MINimum** default of zero causes empty search results to be saved as valid search orders.

The maximum and minimum defaults for each command are outlined in Table 5 below:

Table 5 – Default maxima and minima for the DEFINE order

Command	Default MINimum Value	Default MAXimum Value
Find	1	Not applicable
Display	Not applicable	1000
Print	Not applicable	No limit
MAP	Not applicable	1000
SORT	Not applicable	1000
THESaurus	Not applicable	1000

Currently defined maximums and minimums may be viewed using **DEfine?**

### Syntax

To define maximum limits:

```
DEfine {Find|Display|Print|MAP|SORT|THESaurus}  
      MAXimum={value| "No limit"}
```

To define minimum limits:

```
DEfine Find MINimum=value
```

where *value* is the value to be assigned; only 0 or 1 is accepted.

## Examples

### Example 1:

```
DEfine Display MAXimum=200
```

limits the number of terms that can be displayed at one time to 200.

### Example 2:

```
DEfine SORT MAXimum=50
```

permits fifty records at most to be sorted per search result, thus conserving computing time.

### Example 3:

```
DEfine Print MAXimum=100
```

limits a print request to a maximum of one hundred records.

### Example 4:

```
DEfine Find MINimum=0
```

causes empty search results to be stored in the search history list even when **Find** orders result in zero hits. TRIP acknowledges the request with the message "Empty search results are now accepted."

### Example 5:

```
DEfine Find MINimum=1
```

returns TRIP to the default, and produces the message "Empty search results are now rejected."

### Example 6:

```
DEfine MAP MAXimum=200
```

limits the maximum number of source records from which search terms are extracted to 200.



## DEfine MERGe

### Description

**MERGe** is a qualifier for the **Sort** modifier in a **Show** order. It allows hit records from a clustered database to be combined before being sorted and displayed.

The system default is **NO MERGe**. If a number of records from clustered databases are displayed with **Show Sort**, the records are sorted only *within*, not *across*, each database. The records found in the first database opened are thus sorted and shown before any of the records from the second database.

Use **MERGe** in your output order if you want *all* hit records from *all* databases in a cluster included in the sorting.

If merging is often required, change the default using the **DEfine MERGe** order. **DEfine NO MERGe** returns TRIP to unmerged sorting.

### Syntax

```
DEfine [NO] MERGe
```

### Examples

Consider these orders as given consecutively.

#### Example 1:

```
S=1      <499> BAsE all_demo=alice,carroll  
Show Sort=person
```

This series opens the demonstration databases **Alice** and **Carroll** as the cluster database **All\_Demo**. It then shows the records for all persons in **Alice** sorted alphabetically in ascending order, then all persons in **Carroll**.

```
DEfine MERGe
```

defines the default sort order to include merging.

```
Show Sort=person
```

shows the records for both **Alice** and **Carroll**, sorted alphabetically regardless of their database of origin.

#### Example 2:

```
S=2      <499> BAsE all_carroll=alice,carroll  
Show Sort=person MERGe
```

also displays all records for all persons sorted alphabetically by name, regardless of database. This, however, is a temporary CCL instruction, whose action is limited to this order.

## DEfine Page (TRIPclassic only)

### Description

TRIP's **TTY** (Tele**TY**pe) mode default is **P**Age, where **Show** output is non-continuous. The symbol **NO** overrides the one-page-at-a-time screen display, causing the records of a search result to be displayed without pausing.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

```
DEfine [NO] PAge
```

### Examples

*Example 1:*

```
DEfine PAge
```

scrolls output a single page at a time.

*Example 2:*

```
DEfine NO PAge
```

scrolls output continuously.

## DEfine PCode

*Note:*

*This command is not available for Windows.*

### Description

This command is used to select a translation table for printer output.

### Syntax

```
DEfine PCode=file
```

where *file* is the text file name containing the desired translation table. These files should be stored in the directory pointed to by \$TDBS\_PRC for UNIX.

### Examples

*Example 1:*

```
DEfine PCode=ascii_eng
```

specifies that output on the local printer is to be processed using the translation table stored in the UNIX text file \$TDBS\_PRC/ ASCII\_ENG.PRC.

## DEfine Print

### Description

For more information on this CCL order, refer to the section entitled:  
**DEfine MAXimum/MINimum**



## DEfine PRINTER (UNIX only)

### Description

By default, **Print** output is sent to the printer queue defined by the logical name `$TDBS_PRINT` (UNIX only). Creating a PRN file allows users to direct output to other printer queues and set printer characteristics, such as escape sequences for highlighting. PRN files reside in the directory pointed to by `$TDBS_PRC` (UNIX only), and can be referenced from TRIP via the **DEfine PRINTER** command or used temporarily with the **Print PRINTER** command.

Note:

For UNIX only.

### Syntax

```
DEfine PRINTER=printcontrolfile
```

where *printcontrolfile* is the name of the printer control file to be used.

### Examples

#### Example 1:

```
DEfine PRINTER=LPT1
```

sends the output of later **Print** orders to the printer queue named in the printer control file called `$TDBS_PRC:LPT1.PRN` (UNIX only). Refer to Chapter Twelve, 'Environment Setup' in the *TRIPclassic Manager Guide* for more information regarding the format of a PRN file.

## DEfine REVerse

### Description

By default, records are shown and printed in record number order, that is, the order in which they were added to the database. It is often useful to view the records that were added to the database most recently first.

If the **REVerse** modifier is included in a **Show** or **Print** order, the records are shown in reverse or descending record number order; i.e. those records which were added to the database most recently appear first. The **REVerse** and **SORe** modifiers cannot be used in the same **Show** or **Print** order. Use the **DESCending** modifier instead to reverse the order of a sorted output.

To redefine the output order for the remainder of the search session, use **DEfine REVerse**—the records most recently added to the database (those with the highest record numbers) will always be presented first. To return to the system default of ascending order, use **DEfine NO REVerse**.

**DEfine REVerse DESCending** applies only to output generated by output formats containing the <ris> function. Refer to the section on <RIS> in the 'Output Format Reference Guide' of Chapter Six, 'Output Formats', in the *TRIPclassic Manager Guide* for more information.

### Syntax

```
DEfine {[NO] REVerse|REVerse[NO] DESCending}
```

### Examples

Consider these orders as having been given consecutively:

#### Example 1:

```
S=1      <475> BASe Alice
S=2      <9>    Find speaker=cat
DEfine REVerse
Show Format=2
```

opens the demonstration database **Alice**, finds all records containing the term 'cat' in the **speaker** field and defines the default order to be reversed. It then shows all records located by the **Find** order in reverse order by database record number, with the most recently added record first.

#### Example 2:

If these results are shown again using the output format called '3', which includes the <ris> (record in search) function:

```
Show Format=3
```

the first line of every record will contain the database record number and the number of the record within the search. The first line of every record shown for S=2 is given below:

```
RECORD 139    9 OF 9
RECORD 138    8 OF 9
RECORD 137    7 OF 9
```

```
RECORD 102    6 OF 9
RECORD 101    5 OF 9
RECORD 100    4 OF 9
RECORD 99     3 OF 9
RECORD 98     2 OF 9
RECORD 97     1 OF 9
```

However, if these commands are given:

```
DEfine REVerse NO DESCending
Show Format=2
```

the *database record numbers* (and the rest of the record, of course) will be shown in descending order (reversed), but the *record in search (<ris> numbers* will appear in ascending order, as shown below:

```
RECORD 139    1 OF 9
RECORD 138    2 OF 9
RECORD 137    3 OF 9
RECORD 102    4 OF 9
RECORD 101    5 OF 9
RECORD 100    6 OF 9
RECORD 99     7 OF 9
RECORD 98     8 OF 9
RECORD 97     9 OF 9
```

**DEfine REVerse** restores the definition for full reversed output, where both the highest database record number and highest record in search value appear first in the output.

## DEfine SAve BASE

### Description

When saving a search history as a procedure, TRIP includes the **BASE** command by default as part of the procedure.

When using the same saved search procedure for more than one database, however, you may prefer to manually open a database before running the procedure. The **DEfine SAve NO BASE** order will save all pertinent search orders *except* the **BASE** request.

*Note:*

**DEfine SAve NO BASE** has no effect on the **STOP SAve** command.

To reset the default, use the command **DEfine SAve BASE**.

### Syntax

```
DEfine SAve [NO] BASE
```

### Examples

*Example 1:*

```
S=1      <99>  BASE corr
S=2      <19>  Find tdb
S=3      <12>  Find AND 3rip
S=4      <5>   Find AND trip
SAve myproc
```

TRIP includes the **BASE** command by default when it creates the procedure 'Myproc', and provides the message "Search 4 is saved as MYPROC".

*Example 2:*

```
DEfine SAve NO BASE
S=1      <99>  BASE corr
S=2      <19>  Find tdb
S=3      <12>  Find AND 3rip
S=4      <5>   Find AND trip
SAve myproc2
```

This series will not save **BASE Corr** as part of the procedure 'Myproc2'.



## DEfine SCoPe

*Note:*

*The command **DEfine SCoPe** should not be confused with the modifiers **Find SCoPe** and **UPDate SCoPe**, which both perform different functions within TRIP.*

### Description

TRIP allows a user to define a given set of records within a database or cluster of databases, and limit all subsequent orders to just these records. To limit the scope of subsequent orders to a set of records, these records must be defined as a search result set. The command **DEfine SCoPe S=** can be used repeatedly to limit the search result set after any successful search, and will be recorded in the history window.

The **DEfine SCoPe** order removes the scope limit from subsequent orders and restores the system default, where all records in the database are once again accessible.

The **DEfine SCoPe SDI** option restricts searches to only newly-created or modified records. For more information, refer to the section entitled 'DEfine SCoPe SDI' in this guide.

### Syntax

```
DEfine SCoPe [S=n]
```

where *n* is any valid search result number or zero, implying the most recent search made.

### Examples

*Example 1:*

```
S=1      <475> BASE Alice
S=2      <100> Find R=TO 100
S=3      <100> DEfine SCoPe S=2
S=4      <55>  Find mo#
S=5      <18>  Find mouse
S=6      <18>  DEfine SCoPe S=0
S=7      <18>  Find s#
DEfine SCoPe
```

Example 1, above:

opens database **Alice** (S=1), finds the first one hundred records (S=2), limits all subsequent orders to search number two, which contains records one to one hundred (S=3), finds all the terms in the first one hundred records beginning with 'mo' (S=4), finds all the records in the range one to one hundred containing the word 'mouse' (S=5), (S=6) limits all subsequent orders to the records of the most recent search (S=5), which contains any record number from one to one hundred that includes the word 'mouse', finds all the terms which begin with 's' in the records found by search number five (S=7), and removes all scope limits so that all subsequent orders are applied to all records of the database.

## DEfine SCoPe SDI

### Description

SDI is an acronym for **S**elective **D**issemination of **I**nformation, which is an automated method of searching on TRIP record timestamps or numbers.

SDI uses two markers, one of which stores a record number and the other a time stamp. The first SDI marker acts similarly to a book mark, meaning 'all the records up to and including this marker (record number) have already been seen by this user'. The second marker is more complex, storing both a date and time value meaning 'all records indexed before this timestamp marker have been previously viewed by this user'.

A record's indexing timestamp value is updated whenever the record is indexed (i.e. after being created or modified), but a record is only allocated a *new* record number upon creation. SDI searching on record numbers therefore applies only to new records, and SDI searching on timestamps relates to indexed records. In all other respects the two SDI markers work in the same way.

**DEfine SCoPe SDI** instructs TRIP to 'Find all newly indexed records', whereas **DEfine SCoPe SDI NO UPDate** means 'Find all newly added records'. SDI markers are set using the order:

```
UPDate SDI
```

which, if preceded by a **DEfine SCoPe SDI** order, updates your user profile's SDI timestamp marker with the database indexing date and time. If **DEfine SCoPe SDI** has **NO UPDate** added, your SDI record marker will receive the current highest record number in the database.

The next time you begin an SDI search, TRIP will find only those records that have record numbers higher than the highest record you have reviewed, or that have been indexed *after* the stored date and time.

**UPDate SDI** (without modifiers) operates on the same database as the **DEfine SCoPe SDI** order. If another database has been opened between the two commands, TRIP will provide an error message.

The SDI timestamp can be explicitly updated to any value, if for example, any newly modified or added records since the beginning of the month were required. The SDI record number can likewise be explicitly updated to any value.

The use of SDI has no effect on normal searching. A **DEfine SCoPe SDI** request can be removed in the same way as any other **SCoPe** order by using **DEfine SCoPe**, whether the SDI is updated or not.

### Syntax

```
DEfine SCoPe SDI [NO UPDate]
```

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <99>  DEfine SCoPe SDI
UPDate SDI
```

Example 1, above, opens database **Corr**, begins an **SDI** session, limits the scope to records that have been added or modified (and subsequently indexed) since the last **UPDate SDI**, and updates the user record to indicate that the records just searched will not be included in the next **SDI** run. This order updates the **SDI** timestamp marker with the date and time that **Corr** was last indexed.

*Example 2:*

```
S=1      <475> BAsE Alice
S=2      <475> DEfine SCoPe SDI NO UPDate
UPDate SDI
```

opens database **Alice**, begins an **SDI** session and limits the scope to records whose record numbers are greater than that of the **SDI** record marker (which in this case was zero). It then updates the user record to indicate that the records in the search set will not be included in the next **SDI NO UPDate** run. This order updates the **SDI** record marker with the highest record number in the search set.



## DEfine SORT

### Description

For more information on this CCL order, refer to the section entitled:  
**DEfine MAXimum/MINimum**



## DEfine SPace

### Description

When searching in **Text** and/or **Phrase** fields, a space character between two words implies that the words occur adjacent to one another in the given order.

This default can be changed to make the **SPace** character represent *any* of the Boolean **AND** operators or the proximity operator in a **Find** order. **SPace** definitions can also be limited to certain field types such as **Phrase** or **Text**, named fields (which must be of type **Phrase** or **Text**), named **Views** (collections of fields of type **Phrase** or **Text**) or a combination of these.

Such **DEfine** requests affect only those search instructions containing the same field, field type or view name as specified in the **DEfine** order.

**SPace** can also function as a word mask, an alternative that is at the same time less restrictive than the system default, and more limiting than **AND.S**. Refer to the section entitled 'DEfine MASK' in this guide for more information about word masking.

For a list of all possible Boolean **AND** operators, refer to the section on **Find** presented later in this guide.

*Note:*

*If the space character is redefined to be an operator using the **DEfine SPace** order, that redefinition will be used when searching with the **Fuzz** command.*

### Syntax

```
DEfine SPace [({fieldtype|field|view}
               [{fieldtype|field|view}[,...]])] [= [{AND[.x] | (y..z)}]]
```

where *fieldtype*, *field* and *view* are field types, field names and/or view names to be defined, **AND.x** is any Boolean **AND**. operator, and *y* and *z* are values indicating the allowable number of terms that may appear between two search target terms.

**DEfine SPace** without arguments resets **SPace** to the default.

### Examples

*Example 1:*

These orders,

```
S=1      <475> BAsE Alice
DEfine SPace=AND.F
S=2      <1> Find turtle execution
```

open the demonstration database **Alice** and search for the terms 'turtle' **AND** 'execution' within the same field. They may be separated by any number of terms, and may occur in either order.

*Example 2:*

This command:

```
DEfine SPace=AND.S
```

defines the **SPace** character to represent the operator **AND.S**. If the following two **Find** orders are given immediately after it for database **Alice**, they are equivalent:

```
S=1      <475> BAsE Alice
S=2      <1>   Find tunnel AND.S well
S=3      <1>   Find tunnel well
```

**Example 3:**

```
DEfine SPace=AND.P
```

defines the **SPace** character to represent the operator **AND.P**. If these two **Find** orders were given immediately afterward for **Alice**, they would be equivalent:

```
S=1      <475> BAsE Alice
S=2      <1>   Find (tunnel AND.P well) AND.R (maps
            AND.P pictures)
S=3      <1>   Find (tunnel well) AND.R (maps
            pictures)
```

**Example 4:**

```
S=1      <475> BAsE Alice
DEfine SPace=(-1..0)
S=2      <34> Find Alice was
```

This order defines the **SPace** character to represent the proximity operator, which finds records containing adjacent search terms in the order specified or in reverse order separated by (at most) one term.

The **Find** order would locate phrases such as 'Alice was puzzled' and 'Alice was beginning', as well as 'when that was done," Alice said' and "But it certainly was funny," Alice said'.

**Example 5:**

This series,

```
S=1      <99> BAsE corr
S=2      <0>   DEfine VIEw allnames=rname,sname
DEfine SPace(allnames)=(0..1)
S=3      <47> Find allnames=mats lindquist
```

defines **SPace** to mean 'the given terms must appear in the order specified, with zero or one terms between them', affecting only the **View allnames**.

**Example 6:**

This **DEfine** order,

```
S=1      <99> BAsE corr
DEfine SPace(sname)=AND.S
S=2      <2>   Find sname=rolf larsson
```

defines the **SPace** character to represent the operator **AND.S** only in **Corr**'s **PHrase** field **sname**. The subsequent search finds all records in which both terms 'rolf' and 'larsson' exist in the same subfield of the **PHrase** field **sname**.

*Example 7:*

This **DEfine** series,

```
S=1      <99>  BAsE corr
DEfine SPace(PHrase)=AND.S
S=2      <4>   Find PHrase=rolf larsson
```

defines the **SPace** character to represent the operator **AND.S** in **PHrase** fields only. The subsequent search finds all records in which both 'rolf' and 'larsson' exist in the same subfield of any **PHrase** field.

*Example 8:*

This series,

```
S=1      <99>  BAsE corr
DEfine SPace (rname)=AND.S
S=2      <15>  Find rname=mats lindquist
S=3      <2>   Find mats lindquist
DEfine SPace (PHrase)=AND.S
S=4      <47>  Find PHrase=mats lindquist
S=5      <2>   Find mats lindquist
S=6      <0>   DEfine VView allnames=rname,sname
DEfine SPace (allnames)=AND.S
S=7      <2>   Find mats lindquist
S=8      <15>  Find rname=mats lindquist
S=9      <47>  Find allnames=mats lindquist
```

performs the following operations:

- opens the demonstration database **Corr** (S=1),
- defines the **SPace** character to represent the operator **AND.S**, only in the **PHrase** field **rname**,
- finds all records in which both terms 'mats' and 'lindquist' exist in the same subfield of **rname** (S=2),
- finds all the records in which the term 'mats lindquist' exists in any **PHrase** or **Text** field (S=3),
- defines the **SPace** character to represent the operator **AND.S**, in **PHrase** fields,
- finds all records in which both terms 'mats' and 'lindquist' exist in the same subfield of any **PHrase** field (S=4),
- finds all records in which the term 'mats lindquist' exists in any **PHrase** or **Text** field (S=5),
- defines a **View** called 'Allnames' to contain the **PHrase** fields **rname** and **sname** (S=6),
- defines the **SPace** character to represent the operator **AND.S** in the **View** 'Allnames' (in addition to all **PHrase** fields defined above),

- finds all records in which the term 'mats lindquist' exists in any **PHrase** or **Text** field (S=7),
- finds all records in which both terms 'mats' and 'lindquist' exist in the **rname** field (S=8),
- and finds all records in which both terms 'mats' and 'Lindquist' exist in the same subfield of the fields **rname** and/or **sname** (S=9).

**Note:**

*The **AND.S** operator was defined for the **View**, not for each field individually. It finds all of the records in which both terms 'mats' and 'lindquist' exist in the same subfield of the fields **rname** and/or **sname**.*

**Example 9:**

Even when the system space default has been altered, you can still search for adjacent words and word patterns without redefining **SPace** using **AND**.

```
S=1      <475> BAsE Alice
S=2      <51>  Find red queen
S=3      <54>  Find red AND queen
S=4      <53>  Find red AND.F queen
DEfine SPace=AND.F
S=5      <53>  Find red queen
S=6      <51>  Find red AND. queen
```

In the demonstration database **Alice**, using the system **SPace** default, TRIP finds fifty-one records that contain the adjacent words 'Red Queen', fifty-four records containing 'red' AND 'queen' anywhere in the record and fifty-three records with 'red' AND 'queen' anywhere within the same field.

If **SPace** is then redefined to mean '**AND** within the same field' (or **AND.F**) for the remainder of the search session and the search for the phrase 'Red Queen' is repeated, TRIP finds fifty-three hits—the total for the previous search request 'Find red **AND.F** queen'.

To search properly for 'Red Queen', use 'Find red AND. queen' to specify adjacency. TRIP finds fifty-one hits once again.

**Example 10:**

**SPace** is restored to the system default by any of the following orders:

```
DEfine SPace
DEfine SPace=
DEfine SPace=AND.
```

TRIP provides the message "Space within TEXT+PHRASE is now treated as SPACE".



## DEfine – STEMming

### Description

Enables and disables the stemming feature of the **FUZZ** command when displaying results using the **Display STEMming** order.

### Syntax

```
DEfine [NO] STEMming
```

*Note:*

*This modifier is **off** by default*



## DEfine STop WOrd

### Description

The CCL FUZZ command can be set up to ignore common words to prevent them slowing down searches without improving the results. Such words are referred to as STOP WORDS and TRIP indicates if a stop word has been used in a search by displaying the omitted terms in a warning message.

TRIP uses "adaptive stop-words", i.e. common words based on their occurrence rate in the index. To enable stop-words, you can either include a line in the tdbb.conf file (See the TRIP Manager Administration Guide, Environment Setup, "STOP\_WORDS" section) or use an extension to the DEFINE command.

*Note:*

*If stop-words are enabled, you can still search for them by either enclosing them in double quotation marks or preceding them with a plus sign. See the CCL FUZZ command for further details.*

### Syntax

```
DEfine STop WOrd=x,y
```

Here *x* represents the percentage of records in which a word must occur and *y* represents the average number of occurrences, per record, of the same word, before it becomes a stop word. If the thresholds set by *x* and *y* are exceeded, the word in question will be automatically defined as a stop word.

### Examples

*Example 1:*

```
DEfine STop WOrd=70,2
```

This defines any words to be stop-words if they are present in 70 percent of the records in a database and also occur on an average of two instances in each record.

## DEfine THESaurus

### Description

This order specifies a thesaurus to be used during a search session.

### Syntax

```
DEfine THESaurus=[{field1|fieldtype1}:]  
thesaurus[.{field2|fieldtype2}  
[:{field3|fieldtype3|view}]]
```

The elements of the command are identified in the following table:

**Table 6 – Elements of the order DEfine THESaurus**

Syntax Element	Function
<i>field1</i>	CT and/or UF
<i>fieldtype1</i>	TExt, PHrase, or TExt+PHrase
<i>thesaurus</i>	name of the thesaurus to be opened
<i>field2</i>	CT
<i>fieldtype2</i>	Phrase
<i>field3</i>	the name of any TExt or PHrase field in the target database(s)
<i>fieldtype3</i>	TExt, PHrase, or TExt+PHrase
<i>view</i>	a view across TExt and/or PHrase fields

### Examples

#### Example 1:

```
S=1      <0>  DEfine THESaurus=thesali
```

defines **Thesali** to be the current thesaurus and uses the system defaults: directs searches to the thesaurus **PHrase** fields **CT** and **UF**, extracts search terms from the **CT** field, and searches for all extracted terms in all **PHrase** fields of the database using exact match phrase searching.

#### Example 2:

```
S=2      <0>  DEfine THESaurus=CT:thesali.PHrase:TExt+PHrase
```

defines **Thesali** to be the current thesaurus, directs searches to the **CT** field only, picks terms from all thesaurus **PHrase** fields (**CT** and **UF**), and searches for the terms in all **PHrase** and text fields of the target database.

#### Example 3:

```
S=3      <475> BAsE Alice
```

```
S=4      <0>  DEfine THESaurus=CT:thesali.PHrase:'speaker'
```

specifies that there be exact matching between the search term 'speaker' and the target fields. This order defines **Thesali** to be the current thesaurus, directs searches to the **CT** field only, picks terms from all indexed thesaurus **PHrase** fields (**CT** and **UF**), and searches for the terms in the speaker field of the target database, using exact match phrase searching.

*Example 4:*

```
S=5      <475> BAsE Alice
S=6      <0>   DEfine
THEsaurus=Phrase:thesali.Phrase:Text+PHrase
```

defines **Thesali** to be the current thesaurus, directs searches to the **CT** and **UF** (synonyms) both for searches in all phrase and text fields.

NOTE: For search, the CT and UF fields are by default the only indexed fields in a thesaurus. The NR field may optionally also be included in the index, but DO NOT include the fields BT, NT, and RT in the index since this will cause unpredictable behaviour.



## DEfine TIMEForm

### Description

**TIMEForm** determines the significance of hours, minutes and seconds during searching in **Time** fields. **TIMEForm** is useful in searching elapsed-time measurements, rather than values for time of day.

The initial or system default is **TIMEForm=1**, which makes the *hours* value the most significant. If the time significance has been changed (using a **DEfine TIMEForm** order) to **TIMEForm=2**, the *minutes* value is the most significant, and if **TIMEForm=3** is in effect, the *seconds* are the most significant.

Assume that **timefield** is a field of type **Time**. The following table illustrates the effect of **DEfine TIMEForm** on CCL searching:

Table 7 – Effect on CCL searching of the DEfine TIMEform order

If the CCL Command is:	and TIMEForm=:	then TRIP will look for this value:
Find timefield=3	1	03:00:00
Find timefield=3	2	00:03:00
Find timefield=3	3	00:00:03
Find timefield=3:00	1	03:00:00
Find timefield=3:00	2	00:03:00
Find timefield=3:00	3	00:03:00

Redefining **TIMEForm** has no effect on entering a full format time value, such as 1:59:59.

### Syntax

```
DEfine TIMEForm=n
```

where *n* is 1, 2, or 3.

## DEfine TOday EXPand

### Description

The **DEfine TOday EXPand** command toggles the behaviour of the **TOday()** function. The default behaviour is **NO EXPand**, which shows the **TOday()** function in the search history exactly as entered. The alternate behavior, **EXPand**, shows the calculated date literal in the search history.

### Syntax

```
DEfine TOday [=] [NO] EXPand
```

#### Notes:

- When the command “**DEfine TOday Expand**” is issued, the status message “*TODAY function set to expand to date literals.*” will confirm correct execution.
- When the command “**DEfine TOday NO EXPand**” is issued, the status message “*TODAY function set to not expand to date literals.*” will confirm correct execution.
- For more information regarding this command, please refer to the section in this manual entitled ‘**Find TOday**’.

## DEfine TStamp

### Description

Enables a timestamp search of records, excluding updated records

### Syntax

```
DEfine TStamp [NO] UPDate
```

all sequential timestamp searches will try to exclude updated records by doing a reverse checking of the timestamps in the BAF file. The search is stopped as soon as a timestamp less than the one being searched for is found.



## DEfine UNKnown Field

### Description

Determines how unknown field names shall be treated in FIND and FUZZ expressions.

The intended use case is for applications that search in dynamically assembled runtime clusters of databases with different designs, and for which search expressions are autogenerated and may therefore sometimes refer to fields that do not exist in any of the databases currently open for search.

### Syntax

To set the behaviour to fully ignore any parts of conditions for FIND and FUZZ that refer unknown fields:

```
DEfine UNKnown FField = IGnore
```

To set the behaviour to treat any part of a condition for FIND and FUZZ that refers to an unknown field as if the expression would yield no hits:

```
DEfine UNKnown FField = No Hits
```

To reset the unknown field behaviour to its default, i.e. to cause FIND and FUZZ expressions referring to unknown fields to fail with an error:

```
DEfine UNKnown FField =
```



## DEfine View

### Description

**DEfine View** orders make searching easier by giving groups of **FIELDs** or **VIEWs** a common name.

These orders appear in the search history as search results without hits. To view any currently defined **VIEWs**, use the order **DEfine?**.

### Syntax

```
DEfine VView [view]={view|field
|fieldtype}[, {view|field|fieldtype}
[,...]]
```

where *view* is both the view to be defined, and the name of one or more views to be included in the defined view. *Field* and *fieldtype* are the names of fields and field types to be included in the defined view.

Field views can only comprise field names/views of the same field type, except for **Text** and **Phrase** fields which can be combined.

### Examples

The following examples have been taken from the **Corr** demonstration database provided with TRIP, and are meant to be worked consecutively.

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <0>   DEfine VView vname=rname, sname
```

defines a **View** called **vname** across the **Phrase** fields **rname** and **sname**. If these two orders are given subsequently:

```
S=3      <7>   Find vname=smith
Display vname=sm#
```

TRIP will look only in **rname** and **sname** for the term 'smith' and all terms beginning with 'sm'.

#### Example 2:

If the following two **Views** are defined:

```
S=4      <0>   DEfine VView name=rname, sname
S=5      <0>   DEfine VView country=rcountry, scountry
```

and this order is subsequently given:

```
S=6      <10>  Find name=mats AND country=sweden
```

TRIP will search in the **rname** and **sname** fields for the name 'Mats', and in **rcountry** and **scountry** for the term 'Sweden'.

*Example 3:*

```
S=7      <0>      DEfine VView date=day, moddate
S=8      <23>     Find date < 1984
```

TRIP will search in the **day** and **moddate** fields for dates where the year is less than 1984.

*Example 4:*

If a **VView** called **name** points to the fields **rname** and **sname**, then

```
S=7      <0>      DEfine VView=name, TExt
S=8      <60>     Find mats
```

limits the search for 'Mats' to the **rname** and **sname** fields and any fields of type **TExt**, since no field name was supplied in the search order.

*Example 5:*

```
S=9      <0>      DEfine VView=TExt, PHrase
```

restores the system default of searching in all **TExt** and **PHrase** fields.

## DEfine WEIght

### Description

This order allows the **FUZZ** command to increase the significance of hits in certain fields over others. Valid field weights range from one (the default value) up to the system maximum integer. Practically, however, using values between one and ten will produce reasonable results. All fields are assigned a default weight of one.

When displaying records from a **FUZZ** search, use **Sort=Frequency** to sort them by rank.

*Note:*

*For information on writing a **WEIght** value or record rank using the **<weight>** output format function, refer to Chapter Six, 'Output Formats', in the **TRIPclassic Manager Guide**.*

### Syntax

```
DEfine WEIght(field[,field[,...]])=n
```

where *field* is the fieldname to be weighted, and *n* the weight to be assigned.

*Notes:*

*"DEfine WEIght(...)" only affects currently open databases. It does not affect any currently closed databases that were opened sometime earlier in the current session.*

*When using the fuzz command the default weight for a field is always 1. However, the possibility exists to raise this value in order to make hits in a given field contribute more to the ranking value calculated when searching.*

*It is possible to define the weight for a given field to 0 (zero), to exclude hits in that field from the evaluation of the ranking value.*

*It is recommended that field weights be set no higher than 10; and only then for particularly important fields.*

*It is possible to define field a weight qualified by database name, e.g. "Define weight (DB1.F1, DB2.F2, ...) = 10"*

### Examples

*Example 1:*

This series:

```
S=1      <475> BAsE Alice
DEfine WEIght(person,txt2)=8
S=2      <4>   FUZZ queen hatter alice said
Show SORT=Frequency
```

opens the demonstration database **Alice**,  
makes the **person** and **txt2** fields each eight times more significant than any other field having a weight of one,  
conducts a **FUZZ** search, which locates the following records and occurrences of the search terms:

Record number	Person field <b>Weight=8</b>	Speaker field Weight=1	Txt field Weight=1	Txt2 field <b>Weight=8</b>
113	Queen x 1 Hatter x 1	Hatter x 1	Queen x 1 Hatter x 2 Alice x 2 Said x 2	Alice x 1 Said x 1
114	Queen x 1 Hatter x 1	Hatter x 1	Hatter x 1	Queen x 1 Hatter x 2 Alice x 1 Said x 1
196	Queen x 1 Hatter x 1	<i>No hits!</i>	Queen x 1 Hatter x 1 Alice x 1 Said x 4	<i>No hits!</i>
201	Queen x 1 Hatter x 1	Queen x 1 Hatter x 1	Queen x 2 Hatter x 4 Alice x 1 Said x 3	<i>No hits!</i>

and displays the found records in the ranked order of 114, 201, 113, 196.

## DElete

### Description

**DElete** orders erase searches, saved searches, procedures and saved print orders, or in the case of global deletion orders, records.

When given without qualifiers, the **DElete** order deletes the most recent search.

When used with **S=** followed by a list of numbers and/or number intervals, the searches specified by the numbers are deleted. When used with **S=** followed by **ALL**, all searches in the current search history are deleted. Ranges of searches or saved print orders are defined using **TO**, **FRom** and the comma [,]. Saved searches and procedures are deleted using **SAve** or **PRocedure=**.

Only global deletion orders can delete data. Exercise caution when using these, as they are extremely powerful. Global updates operate only on the **BAse** File (**BAF**) and do not update the indices, therefore indexing must be done separately. Global deletion orders are covered in the next section, entitled 'DElete–Global Orders'.

### DElete – Section Index

**DElete** is discussed under the headings given below:

Table 8 – List of DElete order sections

DElete	Search Result
DElete	Print Request
DElete	Procedure
DElete	Global Update

### Syntax

For search order deletions:

```
DElete [S={range|ALL}]
```

For print order deletions:

```
DElete P={range|ALL}
```

For procedure deletions:

```
DElete {SAve|PRocedure[=]} procedure
```

where **S** stands for *search*, **P** represents *print*, *range* specifies one or a series or range of search or print numbers and *procedure* is the name of a macro or procedure to be deleted.

For global deletion orders:

To delete items contained within records specified by a record number or record number list:

```
DElete [COpy[=database]] {R|field [. {subfieldno|paragraphno  
[.sentenceno]}}}  
Where R=range
```

To delete items contained within records specified by a search result number:

```
DElete [COpy[=database]]  
  {R|PART|FIELD|SUBField|PARagraph|SENTence|Word|  
  field[.{subfieldno|paragraphno  
  [.sentenceno]}} WHERE S=n
```

where R stands for *record*. *Field*, *subfieldno*, *paragraphno* and *sentenceno* are the numbers of the field/subfield/paragraph/sentence whose contents are to be deleted, *range* represents one or more record numbers and *n* is a search result number in which the deletion(s) will occur.



## DElete – Search Result

### Description

This order is used to delete one or more searches. Used alone, it deletes the last search. When followed by **S=** and a number, list of numbers or number interval, it deletes the specified searches, and when used with **S=ALL**, all searches are deleted.

### Syntax

```
DElete [S={range|ALL}]
```

where *range* represents a record number, list of record numbers or range of record numbers.

### Examples

*Example 1:*

```
DElete
```

deletes the most recent search from the search history.

*Example 2:*

```
DElete S=TO 3, 5 TO 7, FRom 9
```

deletes all searches except search numbers four and eight from the search history.

*Example 3:*

```
DElete S=ALL
```

deletes all searches from the search history.

## DElete – Print Request

### Description

This order is used to delete one or more print requests. When followed by **P=** and a number, list of numbers or number interval, it deletes the specified print requests, and when used with **P=ALL**, all print requests are deleted.

### Syntax

```
DElete P={range|ALL}
```

where *range* represents a record number, list of record numbers or range of record numbers.

### Examples

#### Example 1:

```
DElete P=3
```

deletes print order number three from the print queue. Use the command **Print?** to view print jobs on the queue and obtain print order numbers for deletion.

#### Example 2:

```
DElete P=TO 3, 5 TO 7, FRom 9
```

deletes all print orders except search numbers four and eight from the print queue.

#### Example 3:

```
DElete P=ALL
```

deletes all print orders from the print queue.



## DElete – Procedure

### Description

This order is used to delete macros and procedures.

### Syntax

```
DElete {SAve|PRocedure[=]} procedure
```

where *procedure* is the name of the procedure to be deleted.

### Examples

*Example 1:*

```
DElete proc news
```

```
DElete proc=news
```

```
DElete SAve news
```

Each of these orders deletes the TRIP macro or procedure called 'News'.

## DElete – Global Update

### Description

**DElete** is also used as a command for global updating, deleting a particular record, part record, word, sentence, paragraph, field or subfield from all of the specified records of one database simultaneously. Any user of global delete must have write privileges to the database concerned.

In order to avoid deleting the complete content of a database by mistake the global record delete can have a maximum value defined; see the 'Define – **Delete**' section of this manual for more details.

The qualifier **WHere** states which records should be deleted. It is followed by **S=** and a search result number, or **R=** and a record number or list or range of record numbers. Ranges of search result numbers or record numbers can be specified using **TO** and **FRom**, while exact values are specified using equality symbols. Ranges and values are separated by commas.

### Syntax

To delete items contained within records specified by a record number or record number list:

```
DElete [COpy[=database]]{R|field
      [{subfieldno|paragraphno|.sentenceno}]}}
WHere R=range
```

To delete items contained within records specified by a search result number:

```
DElete [COpy[=database]]
      {R|PART|FIELD|SUBField|PARAgraph|SENTence|Word|
      field[.{subfieldno|paragraphno
      [.sentenceno]}]} WHere S=n
```

where *R* stands for *record*. *Field*, *subfieldno*, *paragraphno* and *sentenceno* are the numbers of the field/subfield/paragraph/sentence whose contents are to be deleted, *range* represents one or more record numbers and *n* is a search result number in which the deletion(s) will occur.

### Examples

*Example 1:*

```
DElete R WHere R=TO 3, FRom 98
```

deletes the first three records, record number ninety-eight and all subsequent records of the open database.

*Example 2:*

```
DElete FIELD WHere S=2
```

deletes the contents of the field located by search result number two from the records of that search result.

*Example 3:*

```
DElete SUBField WHere S=7
```

deletes the subfields found by search number seven from the records located by that search result.

*Example 4:*

```
DElete PARagraph WHere S=23
```

deletes the paragraphs located by search result number twenty-three.

*Example 5:*

```
DElete SENTence WHere S=8
```

deletes the sentences found by search number eight from the records located by that search result.

*Example 6:*

```
DElete PART WHere S=0
```

If the most recent search found part records, these will be deleted. If there were no hit part records, nothing will be deleted.

*Example 7:*

```
DElete COpy=corr rname WHere S=2
```

deletes the contents of the field called **rname** from all records found by search result number two, and adds these now-modified records to the specified copy destination database, **Corr**.

*Example 8:*

```
DElete Txt.3.1 WHere S=14
```

removes the first sentence of the third paragraph of every **Text** field named **txt** located by the fourteenth search result.

*Example 9:*

```
DElete rname.4 WHere S=2
```

deletes the fourth subfield of the **Phrase** field **rname** in search number two.

## Display

### Description

Display orders are used to display the vocabulary lists of a database. Items from these lists can then be incorporated directly into search orders.

The default **Display** area is the vocabulary contained within all of the **Text** and **Phrase** fields of a database, which is alterable using **Define View**. To show the vocabulary of selected fields, the **Display** order must include a field name or type (**Text** or **Phrase**) as well as a term. Items may be truncated and characters masked as in search orders, using the symbols [\$ # ! : &].

*Note:*

*See the section on **Define MASK** for details on redefining symbols.*

If a **Display** list is long, use the <↑ **Arrow**> and <↓ **Arrow**> keys and commands **More** and **Back** to navigate screen output.

The terms found by the latest **Display** list may be included in a search order, the method differing depending on whether the list has been presented in TRIP Manager, or TRIPclassic. For more information, see the sections entitled “Display – The TRIPmanager Display List”, or “Display – The TRIPclassic Display List”.

## Display – Section Index

**Display** is discussed under the headings given below:

**Table 9 – List of Display order sections**

Display	The TRIPmanager Display List
Display	The TRIPclassic Display List
Display	A Simple Order
Display	Field Qualifiers
Display	Display BAsE
Display	Enhanced Output
Display	FRequency
Display	Fuzz
Display	Search Results
Display	SOrt on FRequency
Display	Display – STEMming
Display	Thesaurus Modifiers

## Syntax

For general display:

```
Display [Word][{field|fieldtype|view}=]term  
[S=n][SORT=Frequency][Frequency rel-op number]  
[FIELD=f1,f2,...]
```

For fuzzy displays:

```
Display [Word][{field|fieldtype|view}=]FUZZ  
(term [,percentage][,depreciated][,find-terms]  
[,algorithm])  
[S=n][SORT=Frequency][Frequency rel-op number]
```

where *word* (for phrase fields only) displays a list of all words contained in *field*, *fieldtype* or *view*; *term* is the search expression to find, *percentage* is the desired percentage of word matching (range: 1\*100); *depreciated* and *find-terms* have no effect; *algorithm* is the search algorithm to be used (1 or 2) and *n* is a search result number.

To display terms from a thesaurus:

```
Display {CT|BT|NT|RT|UP|DOWN}(term)
```

## Display – The TRIPmanager Display List

### Description

A display list is a list of vocabulary terms that exist (in the fields and subfields specified) in the database. In TRIP Manager, the list appears in a new window, an example of which is shown below:

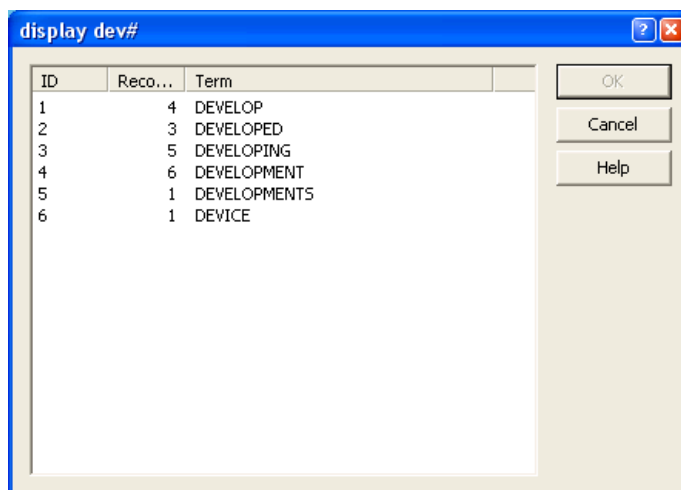


Figure 4 – TRIP Manager Display term list

Each term is numbered and the number of records in which each term is present (not the number of occurrences) is also shown.

A single term number or list or range of term numbers may be used to select terms from a **Display** list to form a **Find** order.

- To select a term, simply click on it with the left mouse button.
- To select multiple terms, hold down the <Ctrl> key whilst left-clicking with the mouse.
- To select blocks of terms, hold down the <Shift> key whilst left-clicking with the mouse.
- Multiple selects and block selects may be combined if required.

When display window's the **OK** button is clicked, the selected terms are transferred to the CCL Query window as a **Find** order.

Alternatively, the terms found by the latest **Display** list may be included in a search order in the CCL Query window, either alone or in combination with other search terms, by incorporating their *term numbers* (T=n) in a CCL **Find** order:

```
Find T=1,3
```

### Examples

The examples shown below are meant for use with the demonstration database **Corr**.

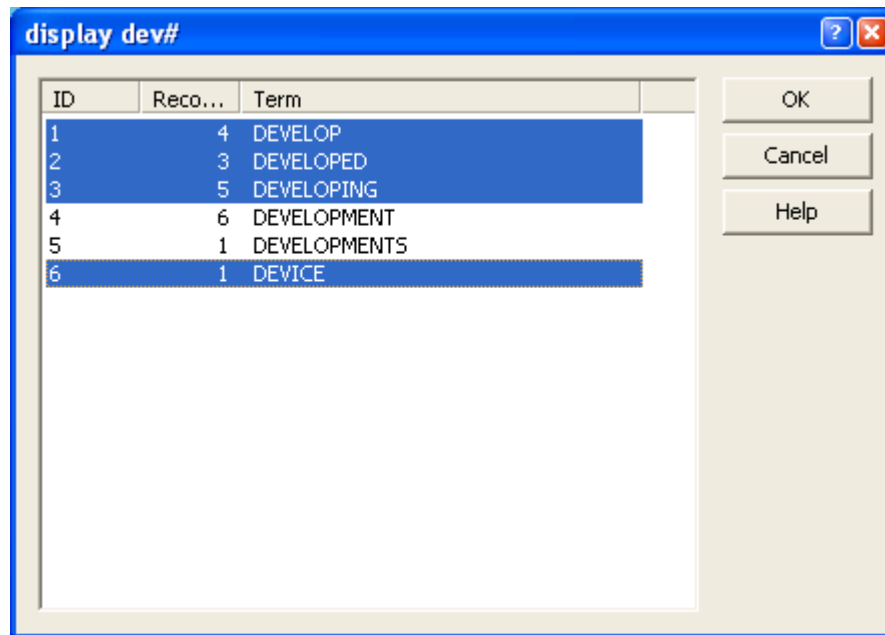
#### Example 1:

```
S=1      <99>  BAsE corr
```

```
Display dev#
```

displays a list of all terms which begin with the stem 'dev' as shown in figure 4 above.

To form a search result from the **Displayed** list, use the mouse to select the desired terms as shown in the figure below and then click on the display window's **OK** button.



**Figure 5 – TRIP Manager Display list with items selected**

For example, selecting the first five terms and clicking on the display window's **OK** button results in the following find order:

```
S=2      <15> Find ("DEVELOP" OR "DEVELOPED" OR "DEVELOPING"  
OR "DEVICE")
```

and locates all records containing the terms 'develop', 'developed', 'developing', or 'device', which in this case would total thirteen records.

## Display – The TRIPclassic Display List

### Description

This is a list of vocabulary terms that exist (in the fields and subfields specified) in the database. In TRIPclassic, the list appears in the search window. Each term (abbreviated **T**) is numbered, and the number of records in which each term is present (not the number of occurrences) is also shown. The total number of terms found is shown at the top of the list.

A single term number or list or range of term numbers may be used to select terms from a **Display** list to form a **Find** order. When using lists and/or ranges, intervals are formed using **TO** and **FRom**, e.g. 'TO 3', '5 TO 7' and 'FRom 10'.

There are three types of intervals: 'TO a number', 'a number TO a number' and 'FRom a number'. In any order that uses a mixture of **TO** and **FRom** intervals, the interval 'TO a number' must always be given first, then 'a number TO a number', and finally 'FRom a number'. Were all three examples to be combined in the same order, they would have to appear in the order given above.

### Display Term Selection

The terms found by the latest **Display** list may be included in a search order, either alone or in combination with other search terms, by incorporating their *term numbers* (T=n) in an order:

```
Find T=1,3
```

or by using **<Select>** to import the terms to the CCL window:

- enter the command **Select** on the command line, or press the **<Select>** key to move the cursor to the display window,
- scroll the list using the **<↑ Arrow>**, **<↓ Arrow>**, **<Next>** and **<Prev>** keys,
- with the cursor beside the desired term, press **<Select>**,
- repeat the **<Select>** step as many times as needed,
- and press **<Enter>** to return to the command window and form the new search result.

### Examples

The examples shown below are meant for use with the demonstration database **Corr**.

#### Example 1:

```
S=1      <99> BASe corr
Display dev#
```

displays a list of all terms which begin with the stem 'dev'. This list could include the following:

```
T=1      <4>  DEVELOP
T=2      <3>  DEVELOPED
T=3      <5>  DEVELOPING
T=4      <6>  DEVELOPMENT
T=5      <1>  DEVELOPMENTS
T=6      <1>  DEVICE
```



'T=' denotes a *term number*, which is assigned to each term after sorting. The numbers within angle brackets [**<** **>**] indicate the number of records that contain the given term (remember that each record may contain more than one instance of each term).

To form a search result from the **Displayed** list, use **Find** and the term number(s) for the terms you wish to include. For example,

```
S=2      <15> Find T=1 TO 5
```

locates all records containing the terms 'develop', 'developed', 'developing', 'development' and 'developments', which in this case would total fifteen records.

**Example 2:**

```
Display sname=#
```

```
Display sname=$
```

The two **Display** orders given above are equivalent. Although the field (**sname**) used in this example is of type **Phrase**, **Text** fields and **Views** can also be **Displayed**. TRIP provides lists of all terms in these categories, provided that the number of terms available does not exceed the current limit dictated by **Define Display MAXimum**.

**Example 3:**

```
S=1      <475>      BASE alice
```

```
Display word person=a#
```

displays a list of all words in the *person* field *beginning* with the letter 'a'.

**Note:**

*Without the Word modifier you will get a display list with all phrases in person containing a word beginning with the letter 'a'.*

## Display – A Simple Order

### Description

A simple display presents the vocabulary around a single term for **PHrase** and **TEText** fields in the database. All the truncation and masking symbols defined for the **Find** command can be used.

For more information regarding truncation and masking symbols, refer to the 'DEfine MASK' and 'Find–Truncation and Masking' sections in this guide.

### Syntax

For general term display

```
Display term
```

where *term* is the term to be displayed.

For subfield specific term display:

```
Display {field|fieldtype|view}[.n][=term]
```

where *fieldname*, *fieldtype* and *view* are the names of fields, field types and views respectively being searched, *n* represents the required subfield number and *term* represents the term to be displayed.

### Examples

*Example 1:*

```
S=1      <99> BAsE corr
```

```
Display $son
```

displays a list of all terms which end in 'son'.

*Example 2:*

```
S=2      <475> BAsE Alice
```

```
Display #qu#
```

displays a list of the terms which contain the string 'qu' anywhere in (not necessarily *within*) them.

*Example 3:*

```
S=1      <475> BAsE corr
```

```
S=2      <1>   Display raddr.2=sparkletown
```

displays a list of the terms in which contain the string 'sparkletown' in the second subfield of the field *raddr*

## Display – Field Qualifiers

### Description

Field qualifiers can be used with the **Display** command in the same way as with the **Find** command. i.e. by field or view name, or field type. Only fields of type **PHrase**, **TExt**, **DAte** or **TIme** are eligible for display and a comma separated list of fields can also be used (See *example 6*).

**Display** operates somewhat differently between **TExt** and **PHrase** fields. For **TExt** fields, only a single term may be displayed at a time, whereas with **PHrase** fields, more than one term is allowed. Thus when no field qualifier is used and the default for **Display** orders is set to all **TExt** and **PHrase** fields, only a single term may be displayed.

When using the name of a **PHrase** field or the field type **PHrase** in a **Display** order, the entire contents of the subfield containing the desired term will be listed; otherwise only term will be given alone.

**DAte** and **TIme** fields or views can be used in **Display** commands in order to obtain a list of values fulfilling a given request.

If TRIP finds a greater number of matches than is allowed by the predefined maximum number for **Display** orders, it stops searching and gives a message to this effect. Either a longer stem must be given, or the maximum number should be increased using the **DEfine** command. Refer to the **DEfine** section covered previously in this guide for more information.

### Syntax

```
Display {field|fieldtype|view}=term
```

where *field*, *fieldtype* and *view* are the names of fields, field types and views whose contents are to be displayed, and *term* is the term to search for within that field, field type or view.

### Examples

The following were taken from the demonstration database **Corr**.

*Example 1:*

**Display** can be used to search for a phrase where certain of the characters are not known. For example, to locate the full names of all persons whose first name begins with 'R' and whose last name is 'Smith', use

```
Display PHrase=R$ Smith
```

to display a list of each subfield within *any* **PHrase** field that matches this pattern. The resulting list will present each unique expression where one term begins with 'R' and the term immediately following is 'Smith' stored in a **PHrase** field of the open database, as well as the number of records in which each variant appears.

*Example 2:*

```
Display rname=Mats L$
```

lists each unique expression beginning with 'Mats L' that is stored in the **PHrase** field **rname**, along with the number of records in which he appears.

*Example 3:*

```
Display content=#tation
```

displays a list of the terms which exist in the **content** field that ends with the letters 'tation'.

*Example 4:*

```
Display rname=$
```

displays all contents of the field **rname**.

*Example 5:*

```
Display day=1984
```

will return a list of all 'day' field values for the year 1984.

*Example 6:*

```
DISPLAY f1,f2,f3=xyz
```

will return a list of fields containing 'xyz'.

## Display BAsE

### Description

Displays search results for individual databases in a cluster

### Syntax

```
Display BAsE=dbname1,dbname2,dbname3...
```

### Examples

#### Example 1:

```
Base x=alice,carroll,corr
Find the
Display base
```

This returns a term list as:

```
T=1 <461> ALICE
T=2 <24> CARROLL
T=3 <87> CORR
```

It is now also possible to select "terms" from this list, e.g.:

```
Find t=2,3
```

#### Example 2:

Base lists can be shown using various methods:

```
Display base=#
Display base=alice
Display base=corr,carroll
```

#### Example 3:

It is also possible to search using the BASE operator, e.g.:

```
Find base=alice,corr
```

This will return a union of the records in ALICE and CORR

## Display CLASS()

### Description

This function displays records that have been assigned category tags, and can be combined within any Boolean expression as normal.

The 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 that 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 found (i.e. record IDs) are then located within the open database.

See the separate document [TRIP\\_White\\_paper\\_Classification.pdf](#), Appendix B of the TRIPmanager Administration guide and the TRIPmanager help file for an in-depth description of document classification in TRIP.

#### Note:

*Before the Class() function can be used, a Classification schema must be created and the databases to be searched must be enabled for category search. Both of these activities are done using the TRIPmanager administration program.*

### Syntax

```
Display Class(name)
```

where *name* is a category name in a classification scheme setup via TRIPmanager.

### Examples

#### Example1:

```
Display class(economic)
```

## Display – Enhanced Output

### Description

Displays the content of a given field, alongside the output from up to 10 tupled fields

### Syntax

```
Display f1=abc# field=f2,f3,...
```

### Example

#### Example 1:

```
Base corr
```

```
Display rname=mats FIeld=raddr, rcountry
```

This returns the following output:

```
T=1      Mats G.           -> Box 2284; Sverige
          Lindquist
T=2      Mats Lindquist    -> Box 2284; Sverige
T=3      Mats Löfström     -> 103 17 STOCKHOLM
T=4      Mr. Mats         -> Box 2284; Sverige
          Löfström
```

#### Note:

*This only works in TRIPclassic or applications using reports (also known as output formats).*

## Display – FRequency

### Description

Displays terms based on occurrence counts by using the **FRequency** modifier.

*Note:*

*This command is only guaranteed to work correctly when searching single databases.*

### Syntax

```
Display field=$ FRequency > n
```

where *field* is the name of a field whose contents are to be displayed, and *n* is required **FRequency** value.

### Examples

#### Example 1

```
Display person=$ freq > 30
```

displays the following term list of any 'person' field entries that occur more than 30 times

T=1	<32>	HUMPTY DUMPTY
T=2	<38>	KING OF HEARTS
T=3	<34>	QUEEN OF HEARTS
T=4	<50>	RED QUEEN
T=5	<53>	WHITE QUEEN
T=6	<34>	WHITE RABBIT



## Display FUZZ()

*Note:*

The **FUZZ()** modifier to this **Display** command is not to be confused with the CCL **FUZZ** command.

### Description

Fuzzy searching implies searching for similarity, a capability that can prove quite useful when looking for differently spelled or frequently misspelled terms and using **Display FUZZ()** will produce a list of terms similar to the term list given by a simple **Display** order.

In addition to the search term, the **FUZZ()** modifier can take four optional comma separated parameters that alter the performance of the fuzzy search.

### Parameters

1. The first parameter is an integer dictating the percentage similarity any search result has to the original search term. The range of permissible values for this parameter is from 1 to 100.

Default Value: 75

2. The second parameter is depreciated. It has no effect in normal fuzz searching and can be left out; however, the leading comma must be used if any following parameters are entered.
3. The third parameter has no effect in a **Display FUZZ()** command.

*Note:*

See the **Find FUZZ** command for more details on parameter number three.

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

Default=1

*Note:*

The default is also the recommended search algorithm.

### Syntax

```
Display [{field|fieldtype|view}=]FUZZ
(term[,param1][,param2][,param3][,param4])
```

where *field*, *fieldtype* and *view* are the names of fields, field types and views whose contents are to be displayed, *term* is the search term to be matched, and *param1* through *param4* are the optional parameters mentioned above.

*Notes:*

- **Display FUZZ()** can also be used with **S=** and **SORT=Frequency**.
- When any leading parameters are omitted, the separating commas between remaining trailing parameters must still be included.

- Refer to the section '**DEfine FUZZ**' in this manual for more information on redefining the default values of the above parameters, and '**Find FUZZ**' for help with fuzzy searching in general.

## Examples

The following examples are for use with the demonstration database **Alice**.

### Example 1:

```
S=1      <99>  BASE alice
Display FUZZ(late)
```

produces a list such as:

T=1	<10>	LATE
T=2	<2>	LATER
T=3	<15>	RATE
T=4	<4>	SLATE
T=5	<3>	GATE
T=6	<3>	HATE
T=7	<3>	PLATE
T=8	<3>	ATE

The list is sorted on similarity, that is, the most similar search results are nearer the top.

### Example 2:

```
Display FUZZ(late,84)
```

displays terms that correspond to at least an eighty-four percent similarity rating compared to 'late' and produces a list such as:

T=1	<10>	LATE
T=2	<2>	LATER
T=3	<4>	SLATE
T=4	<3>	PLATE

## Display – Search Results

### Description

A **Display** order can be confined to the result set of any previous search simply by including a reference to this search. The list of terms then becomes limited to only those which match the given term, and appear in the result set specified.

### Syntax

```
Display term S=n
```

where *term* is the term to display, and *n* is the search result whose contents are to be displayed. Display orders of previous search results can also include a field name, field type or view name, followed by an equality symbol [=] and the term to be displayed.

### Examples

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <23>  Find produ#
S=3      <9>   Find S=2 AND soft#
Display S=2 produc#
```

presents a list of all terms beginning with 'produc' in the records located by search number two, such as 'produced', 'product', 'produce', 'production' and 'products'.

#### Example 2:

```
S=1      <475> BAsE Alice
S=2      <63>  Find red
S=3      <131> Find queen
S=4      <140> Find s=2 OR s=3
S=5      <68>  Find S=3 and white
Display S=5 speaker=d#
```

lists all terms beginning with 'd' from **Alice's speaker** field in the records collected by search number five.

#### Example 3:

```
S=1      <475> BAsE Alice
S=2      <63>  Find red
S=3      <131> Find queen
S=4      <140> Find s=2 OR s=3
Display person=# S=4
```

This series displays all of the subfields of the **Alice** field **person** for the records of search number four.

## Display – SOrt in REVerse order

### Description

The default sort order for a display list is alphabetical, ascending. This may be changed by adding the qualifier **SOrt=Frequency**, where the term occurring in descending order of frequency.

Sorting can also be done in reversed order by using the qualifier **SOrt=REVerse** for alphabetical descending order and **SOrt=Frequency, REVerse** for frequency ascending order.

### Syntax

```
Display term SOrt=modifier
```

Where *modifier* can be REVerse (or REVerse,Text or TExt,REVerse) for reversed alphabetical order and Frequency, REVerse (or REVerse, FFrequency) for reversed frequency order.

Where term is the term to be displayed. Display orders using **SOrt=** can also include a field name, field type or view name, followed by an equality symbol [=] and the term to be displayed.

### Examples

The examples below are to be used with the demonstration database **Corr**.

#### Example 1:

```
S=1      <99>  BAsE corr
```

```
Display dev# SOrt=Frequency,Reverse
```

results in the following display list:

```
T=1 <1>  DEVICE
T=2 <1>  DEVELOPMENTS
T=3 <3>  DEVELOPED
T=4 <4>  DEVELOP
T=5 <5>  DEVELOPING
T=6 <6>  DEVELOPMENT
```

#### Example 2:

```
Display SOrt=Reverse rname=mats
```

produces this list of terms:

```
T=1 <1>  MR MATS LÖFSTRÖM
T=2 <1>  MATS LINDQUIST
T=3 <4>  MATS LÖFSTRÖM
T=4 <14> MATS G LINDQUIST
```

## Display – SOrt on FRequency

### Description

The default sort order for a display list is alphabetical. This may be changed by adding the qualifier **SOrt=FRequency**, where the term occurring in the greatest number of records appears at the top of the list and the others in descending order of frequency.

The terms in the list are always numbered from one starting at the top, and terms having the same frequency will be sorted alphabetically. As many terms as may be displayed may be sorted in this way.

### Syntax

```
Display term SOrt=FRequency
```

where *term* is the term to be displayed. Display orders using **SOrt=FRequency** can also include a field name, field type or view name, followed by an equality symbol [=] and the term to be displayed.

### Examples

The examples below are to be used with the demonstration database **Corr**.

#### Example 1:

```
S=1      <99>  BAsE corr
Display dev# SOrt=FRequency
```

results in the following display list:

T=1	<6>	DEVELOPMENT
T=2	<5>	DEVELOPING
T=3	<4>	DEVELOP
T=4	<3>	DEVELOPED
T=5	<1>	DEVELOPMENTS
T=6	<1>	DEVICE

#### Example 2:

```
Display SOrt=FRequency rname=mats
```

produces this list of terms:

T=1	<14>	MATS G LINDQUIST
T=2	<4>	MATS LÖFSTRÖM
T=3	<1>	MATS LINDQUIST
T=4	<1>	MR MATS LÖFSTRÖM

## Display – STEMming

### Description

Allows the creation of term lists of all stemmed words found in a field.

### Syntax

```
Display STEMming f1=#
```

where *f1* is the name of the field being displayed and # is the 'wildcard' search..

Notes:

- *Uses Porter Stemmer by default.*
- *Non-Boolean searching is possible across a cluster of different language databases, with appropriate stemming; however, the databases' "Language" settings must have been configured with TRIPmanager: See the **TRIPmanager Administration guide** for more details on how to do this.*
- *For more general information on non\_Boolean searching in TRIP, see the white paper entitled, "TRIP\_White\_Paper\_Non\_Boolean\_Search", which is included in the TRIPsystem distribution's 'doc' directory.*
- *Any databases with no "Language" setting will use the default language value for TRIP, defined in the tdb.conf file.*

### Examples

Note:

*The following example assumes that, before trying it, the field RNAME has been enabled for Non-Boolean searching and the database CORR has subsequently been reindexed.*

Example 1:

The command sequence:

```
S=1      <99>      BASe corr
Display STEMming content=#
```

on a version of the database *Corr* that has non-Boolean searching enabled on the *content* field, will result in a terms list beginning with the following eleven items:

```
T=1      <1>      3RIP
T=2      <1>      ADAM
T=3      <1>      ANDERSSON
T=4      <4>      ARTURO
T=5      <1>      ASGRIMSDOTTIR
T=6      <1>      BILL
T=7      <1>      BOHAN
T=8      <1>      BROWN
T=9      <1>      BUY
T=10     <1>      CHENG
```

## Display – Thesaurus Modifiers

### Description

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 **Display** order can be used to determine the structure and browse the contents of a thesaurus.

#### Thesaurus Structure

Each thesaurus record contains the following elements and descriptions:

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

#### 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 **CTs** and Synonyms. **BTs**, **NTs** and **RTs** 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 **BTs** 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.

#### 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.

#### 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.

For more information, refer to the Chapter Six, 'Thesaurus Searching' of the *TRIPclassic User Guide*.

## Syntax

```
Display {CT|BT|NT|RT|UP|DOWN} (term)
```

where *term* is the term to be searched for in the current thesaurus.

## Examples

The examples in this section use the thesaurus for demonstration database **Alice** called **Thesali**.

### Example 1:

```
S=1      <0>  DEfine THESaurus=thesali
Display CT(mouse)
```

defines the thesaurus as **Thesali** and lists all those thesaurus records that contain 'mouse' in a **CT** or **UF** element:

```
T=1      Mouse
RT:      White Rabbit
```

### Example 2:

```
Display NT(footmen)
```

displays the thesaurus records where a **PHrase** subfield contains 'footmen':

```
T=1      Footmen
T=2      Frog Footman
RT:      Fish Footman
T=3      Fish Footman
RT:      Frog Footman
```

### Example 3:

```
Display UP(alice's friends)
```

displays the thesaurus records with **PHrase** fields containing the successive parent terms of 'alice's friends':

```
T=1      Persons
T=2      Alice's Family and Friends
T=3      Alice's Friends
```

### Example 4:

```
Display RT(dodo)
```

displays the thesaurus records with the term 'dodo' in the **CT** field, as well as the records with **CT** contents matched by the **RT** terms in the first group of records:

```
T=1      Dodo
RT:      Pigeon
COM:      The Dodo, being a fat and slow-moving bird, is
extinct since the seventeenth century. Before that it lived
on Mauritius.
T=2      Pigeon
RT:      Dodo
```



**Example 5:**

```
Display DOWN(alice's friends)
```

displays the thesaurus records with **PHrase** fields containing the successive child terms of 'alice's friends':

```
T=1      Alice's Friends
T=2      Mabel
          RT:  Ada
T=3      Ada
          RT:  Mabel
```



## Edit (TRIPclassic only)

### Description

This order, when given during a search session, allows records from the currently open database to be modified using a data entry form. The user must have write privileges to the database in question, and the database must have a default data entry form associated with it.

When used alone (without modifiers) while browsing the output generated by a **Show** order, **EDit** imports the record currently being displayed into the data entry form. Pressing <Enter> saves the edited record, <Gold><N> presents the next record of the database for editing (without saving the current record), and <Gold><P> brings up the previous record. <Leave> exits data entry without saving and returns you to the **Show** window.

**EDit** followed by **S=** and the number of a search result brings each record of that search result into the data entry form in sequence. Press <Gold><N> to proceed to the next record of the search result to be edited, and <Gold><P> to move to the previous record.

**EDit PART S=** allows only those parts that are hit to be modified. To return to editing a search after leaving the entry form, use **Continue EDit**.

The **COPy** modifier is used in **EDit** orders when copying records from within a database or from one database to another (databases must be compatible). **COPy** also uses a data entry form and must appear directly after the command, e.g. **EDit COPy**. Copy destination may be given using either a **DEfine** order or directly in the editing or updating order, and when the copy destination is also the database of origin or copy source, the new records are appended to the database.

See the 'DELeTe' and 'UPDate' sections of this manual for further information regarding **COPy**.

*Note:*

*This command is valid in TRIPclassic only.*

## Edit – Section Index

**EDit** is discussed under the headings given below:

Table 10 – List of **EDit** orders

EDit	A Simple Order
EDit	Search Result
EDit	INSert
EDit	PART
EDit	COPy
EDit	SORT

### Syntax

To create new records:

```
EDit INSert[BASe=database]
```

To copy and/or modify existing records:

```
EDit [{PART|COpy[=database]]  
[S=n[SORt=field[, field[, ...]]]]
```

where *database* is the name of a destination database, *n* is the search result number whose records are to be edited and *field* is the name of the field(s) on which to sort.



## Edit – A Simple Order

### Description

This order imports the active record in the **Show** window into a data entry form for modification. The default active or current record is the one that is positioned at the bottom of the **Show** window.

To select another record in the **Show** window for editing, type 'se' in the command window (the **SElect** command) and press ↵, or use the <**Select**> key to move the cursor from the CCL command window to the **Show** window.

Use the <**Next**>, <**Prev**>, <↑ **Arrow**> and <↓ **Arrow**> keys to move about the display, and <**Space**>, <**Enter**> or <**Return**> to select a record and return to the command window. The record the cursor was positioned beside or in then becomes the current record.

TRIP uses the database default entry form for editing. If the database designer has not specified a default entry form, one must be defined using **DEfine EForm=entryform**. Refer to the 'DEfine EForm' section in this manual for further information.

### Syntax

EDit

### Examples

#### Example 1:

```
S=1      <99>  BASe corr
S=2      <19>  Find tdbS
S=3      <1>   Find S=2 AND november
Show
EDit
```

The **EDit** command imports the current record (S=3, database record number one) into **Corr**'s default data entry form for editing.

## Edit – Search Result

### Description

This command allows several records of a search result to be edited. Press **<Gold><N>** to reach the next record of the search result, and **<Gold><P>** to bring up the previous record.

### Syntax

```
EDit S=n
```

where *n* is the search result number whose records are to be edited. **EDit S=** can be used with the **PART**, **COPY** and **Sort** modifiers.

### Examples

#### Example 1:

```
S=1      <99>  BAsE corr  
S=2      <19>  Find tdbS  
S=3      <21>  Find or trip  
EDit S=3
```

opens the records of search result number three for editing.

## Edit – INsert

### Description

This command brings up the default data entry form for entry of new records.

### Syntax

```
EDit INsert [BASe=database]
```

where *database* is the name of the receiving or destination database.

### Examples

The examples below are for use with the demonstration database **Corr**.

#### Example 1:

```
S=1      <99>  BASe corr  
EDit INsert
```

will add a new record to whatever database is currently open (in this instance, **Corr**).

#### Example 2:

```
EDit INsert BASe=corr
```

allows the user to enter a form and add new records to the database specified, in this case, **Corr**.

## Edit – PART

### Description

When used without modifiers, **EDit PART** allows one selected part record to be edited. To select a part in the **Show** window for editing, use the **<Select>** key (or type **SElect** in the command window and press ↵) to move the cursor from the CCL command window to the **Show** window.

Use the **<Next>**, **<Prev>**, **<↑ Arrow>** and **<↓ Arrow>** keys to position the cursor beside the desired part record, and **<Space>**, **<Enter>** or **<Return>** to select that part and return to the command window. The record the cursor was positioned beside or in then becomes the current part record.

When used with the **S=** modifier, this command brings only hit part records into the default data entry form for modification.

### Syntax

```
EDit PART
```

**EDit PART** can also be used with the **S=** and **SORT=** modifiers.

### Examples

The examples below are for use with the demonstration database **Carroll**, and should be worked consecutively.

#### Example 1:

```
S=1      <24>  BAsE carroll
DEfine EForm=1
S=2      <2>   Find mome
Show
SElect (select the desired part)
EDit PART
```

brings only parts hit during the last search performed (S=2) into the newly-defined default data entry form (1) for browsing and editing.

#### Example 2:

```
S=3      <1>   Find speaker=unicorn
EDit PART S=0
```

brings only parts hit during the last search performed (S=3) into the default data entry form for browsing and editing.

#### Example 3:

```
S=4      <2>   Find speaker=gryphon
EDit PART SORT=person S=4
```

sorts the records found during the last search performed (S=4) according to the content of the head field **person**, and brings only hit parts into the default data entry form for browsing and editing.

## Edit – COPY

### Description

This command imports records into a data entry form for modification, then appends them to a specified database.

### Syntax

```
EDit COPY[=database]
```

where *database* is the name of the receiving or destination database. This command can also be combined with the **S=** and **SORT=** modifiers.

### Examples

#### Example 1:

```
S=1      BAsE corr
S=2      Find trip
S=3      Find and tdbS
DEfine COPY=corr
EDit COPY S=3
```

makes the database **Corr** the destination of subsequent **EDit COPY** orders using **DEfine**, then imports the records of search result number three into a data entry form for **EDit COPY**ing into destination database **Corr**.

#### Example 2:

```
EDit COPY=corr S=3
```

makes **Corr** the destination database and copies the records located by search result number three into it, as illustrated in the previous example.



## Edit – SORT

### Description

This command sorts the records located by a specified search result before importing them into a data entry form for modification.

### Syntax

```
EDit SORT=field[,field[,...]] S=n
```

where *field* is the name of a field on which to sort, and *n* is a search result number.

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr  
S=2      <83>  Find 3rip  
S=3      <6>   Find and trip  
EDit S=3 SORT=rname
```

will allow the records of search result three to be edited and the records sorted on the contents of the field called **rname**.

## Expand

### Description

**Expand** (or **<Gold><X>**) is used to display more of the surroundings of a hit term when viewing a record in a focused format, displaying all of the current record in the default format rather than just the hit paragraphs or subfields. It can only be used if **FOCUS** is defined as the default, or if the most recent **Show** order contained the **FOCUS** modifier.

Following an **Expand** order, use **Continue** to move to the next focused position, or use **Continue Show** to return to the expanded focus point.

Refer to the 'Show FOCUS' and 'Define FOCUS' sections in this guide for more information.

### Syntax

Expand

### Examples

#### Example 1:

```
S=1      <475> BASe Alice
S=2      <31>  Find TExt=rabbit
DEfine FOCUS
Show
Expand
```

This series opens demonstration database **Alice**, finds all occurrences of 'rabbit' in the **TEText** fields and shows all subsequent search results in **FOCUS**, in which only the sentences or subfields containing the hit terms are displayed. **Expand** temporarily removes the **FOCUS** and shows the current record in its entirety. You will be unable to view any other records until you give a new **Show** order, or use the **Continue** command.

## EXPORT

### Description

**EXPORT** is used to extract the contents of a 'Control' record and write it into a text file, and its companion command **IMPORT** creates or updates a 'Control' record using the contents of this file. This allows database, form and format designs to be copied from one 'Control' file to another. Only the owner/creator of the item to be **EXPORT**ed (database design, form, format etc.) may **EXPORT** it.

Format and entry form names consist of the name of their parent database, a full stop or period and the name of the format itself, for example, **Corr.corr\_entry** or **Alice.1**.

Refer to the section on 'IMPORT' in this manual for related information.

### Syntax

```
EXPORT {BASE|Format|EForm|SForm|THES|PRocedure}=item [PATH]  
FILE=file
```

where *item* is the name of the **BASE**, **Format**, **EForm**, **SForm**, **THESaurus** or **PRocedure** to be exported, **PATH** includes the database path in the export file and *file* is the name of the temporary holding file used to store the exported information.

*Item* can also be *base.\**, which specifies that all database elements (database design, output formats and data entry forms) be exported.

Arguments to a procedure (accessed inside a procedure using %1 to %9) can also be concatenated to form a resultant string, to be used for other CCL commands embedded within the procedure. See example 8 for details.

### Examples

#### Example 1:

```
EXPORT BASE=Alice FILE=alice.def  
IMPORT BASE=wonder FILE=alice.def
```

exports the database design for the demonstration database **Alice** into a file called 'Alice.def', then imports it to a database named **Wonder**.

#### Example 2:

```
EXPORT BASE=Alice FILE=alice.def  
IMPORT UPDATE BASE=mirror FILE=alice.def
```

exports the database design for the demonstration database **Alice** into a file called 'Alice.def', then updates the database design for **Mirror**.

#### Example 3:

```
EXPORT Format=alice.1 FILE=alice.fmt  
IMPORT Format=wonder.1 FILE=alice.fmt
```

exports a format called '1' for database **Alice** into a file called 'Alice.fmt', then imports it to a format named '1' for database **Wonder**.

#### Example 4:

```
EXPORT PRocedure=oldproc FILE=transfer.pro
```

```
IMPort PProcedure=newproc FILE=transfer.pro
```

exports a procedure called 'Oldproc' into a file called 'Transfer.pro', then imports it to a procedure named 'Newproc'.

*Example 5:*

```
EXPORT EForm=alice.1 FILE=alice.ef1
```

```
IMPort EForm=new_alice.1 FILE=alice.ef1
```

exports a data entry form called '1' for database **Alice** into a file called 'Alice.ef1', then imports it to a data entry form named '1' for database **New\_alice**.

*Example 6:*

```
EXPORT BAsE=Alice.* FILE=alice.def
```

```
IMPort BAsE=alice_copy.* FILE=alice.def
```

exports all elements of the database **Alice** (database design, formats and entry forms) and imports them to a database called **Alice\_copy**.

*Example 7:*

```
EXPORT SForm=alice_demo FILE=alice_demo.sfo
```

```
IMPort SForm=alice_new FILE=alice_demo.sfo
```

exports the search form called 'Alice\_demo' to a file called 'Alice\_demo.sfo', and imports it to a new search form called 'Alice\_new'.

*Example 8:*

Saving the following command in a procedure named EXPO\_FRM:

```
EXPORT format=%1.%2 FILE=%3_xyz.frm
```

and calling the saved procedure with

```
RUN EXPO_FRM alice,myform,rabbit
```

will result in the CCL order

```
EXPORT Format=alice.myform FILE=rabbit_xyz.frm
```

## Find

### Description

**Find** is used to build *search orders*, which combine textual or numerical *search terms* with Boolean operators in a statement that summarizes whatever is to be searched for. After making a **Find** request, TRIP creates a search result, assigns it a number and writes a summary line giving the total number of records found.

A search term is a *character string*, one or more words, word fractions or digits. A search term may also be preceded by a *field specification* positioned before the character string, which consists of a search target field name or field type followed by an equality symbol [= <= >= <>]. If no such target is specified, TRIP will automatically conduct all searches in fields of type **Text** or **Phrase**. For information on changing this search default, refer to the section on 'DEfine Vlew' in this manual.

Search terms are combined using the Boolean or logical operators **AND**, **OR**, **XOR** and **NOT**. Search orders may also include a reference to an earlier search result, using **S=** and the search result number rather than an explicit search term.

**Non-Boolean** search can be made using the search function **ABout()** as an operand. Such search expressions can also be combined with other operands using Boolean operators.

If there are records in the database which have been added since it was last indexed, these will not be found by content searching methods unless they have a record name. Only searches on the record number and timestamp will yield accurate results under these circumstances.

## Find – Section Index

As **Find** is the most complex command and can be broken down according to the main types of expressions used, the variants of **Find** are covered by subject rather than by command or subcommand:

Table 11 – List of FIND order variants

Category	Topic
Overview	Find – A Simple Order
Logical Operators	AND operator
	OR operator
	XOR operator
	NOT operator
Direct Searching: Text	Find within fields
	Find if Field Content Exists
	Truncation and Masking
	Proximity Searching
	Previous Searches
	Find Terms From a Display List
	Order of Operator Precedence
	Find FUZZ
Direct Searching: Other	Relational Operators
	Find Record number
	Find TStamp
	Find DUPLICATE
	Find User
Indirect Searching: Text	Find THESaurus
	Mapped Transaction Sets
	Internal Transaction Sets
	External Transaction Sets
Non-Boolean Searching	Find ABout
Category Searching	Find Class
	Find SCope

Use **Find?** to obtain a complete list of all search results compiled during the current search session.

## Syntax

```
Find [search expression] [{AND|OR|XOR|NOT}
    search expression[...]]
```

Here *search expression* can be one or more terms to be searched for in the default **Text** and **Phrase** field search area, or in a specific field, field type or view. *Search expression* can also be a record number or a range of record numbers, a reference to one or more previous search results or terms from a **Display** list, a **FUZZy** search or an indirect search. The various options are discussed within their respective **Find** subsection.

## Find – A Simple Order

### Description

This is the simplest and most straightforward type of search.

**Find**, when used alone, searches for every indexed record in the database. When used with a search term, **Find** searches for the given term by default in all **Text** and **Phrase** fields of the open database.

**Find fieldname.n** (where **n** represents a specific subfield number of the field *fieldname*) restricts the find to the specified subfield.

**Find?** displays a list of all searches made during the current search session. Use the **More** and **Back** commands or the <↑ **Arrow**> and <↓ **Arrow**> keys to scroll the list.

### Syntax (a)

For general searching:

```
Find [term]
```

where *term* represents the search expression to find.

### Syntax (b)

For subfield specific searching:

```
Find fieldname.n=[term]
```

where *fieldname* represents the name of the field being searched, *n* represents the required subfield number and *term* represents the search expression to find.

### Syntax (c)

For key value pair (kvp) specific searching:

```
Find key=value
```

where *key* represents the key of the kvp being searched and *value* represents the value to find.

*Note:*

*This will work only with a kvp defined. See the Define – **KVP** section of this reference for more details.*

### Syntax (d)

To display the current search history:

```
Find[?]
```

### Examples

*Example 1:*

```
S=1      <475> BAsE Alice
```

```
S=2      <475> Find
```

finds all of the records in database **Alice**.

**Example 2:**

```
S=1      <475> BAsE Alice
S=2      <1>   Find apple
```

finds the records in which the term 'apple' exists in any field of type **PHrase** or **TExt**.

**Example 3:**

```
Find?
```

brings the search history window into view, and displays the earliest search performed first.

**Example 4:**

```
S=1      <475> BAsE corr
S=2      <1>   Find raddr.2=sparkletown
```

finds the records in which the term 'sparkletown' exists in the second subfield of the field *raddr*.

**Example 5:**

```
S=1      <99>      BAsE corr
```

after issuing the define command:

```
define kvp=rname,rcountry
S=2      <4>      Find mats=sweden
```

finds the records in which the term 'mats' exists in the *rname* field *and* the kvp 'sweden' exists in the *rcountry* field.



## Find – AND operator

### Description

All of the **AND** operators require that two search terms or expressions must occur in the same record. Each individual **AND** then also signifies further restrictions. The 'ANDs' are outlined below:

Table 12 – List of FIND AND operators

Operator	Meaning
AND	adds one search condition to another; finds records that contain two or more terms anywhere within the record. AND is treated as AND.E by default when used with head and part records.
AND.	a space between words when SPace has been redefined
AND.C	'AND within the same record component' (for use with record heads and parts)
AND.E	'AND within the same record entity' (for use with record heads and parts)
AND.F	AND within the same field
AND.N[n]	AND.'NEAR' within optionally specified distance (n) and independent of order; where $0 \leq n \leq 255$ . Default is 0
AND.P	'AND within the same TEXT paragraph or PHrase field'
AND.R	'AND within the same record' (for use with record heads and parts)
AND.S	'AND within the same TEXT sentence or PHrase subfield'
AND.T	'AND from the same subfield number', i.e. within the same data tuple
AND.W	'AND within the same word'

Refer to the section called 'DEfine AND' in this manual for information about defining the head-part operators **AND.C**, **AND.E** and **AND.R**.

### Syntax

```
Find [search expression] AND[. [x]] search expression
[AND[. [x]] search expression [...]]
```

where *search expression* can be one or more terms to be searched for in the default **Text** and **PHrase** field search area, or in a specific field, field type or view. *Search expression* can also be a record number or a range of record numbers, a reference to one or more previous search results or terms from a **Display** list, a **FUZZy** search or an indirect search. 'AND.x' is the desired **AND** operator.

## Examples

These examples are taken from demonstration database **Carroll**.

### Example 1:

```
S=1      <24>  BAsE carroll
S=2      <1>   Find grass AND tree
```

finds records that contain both terms 'grass' and 'tree' anywhere within the same record.

### Example 2:

```
S=3      <13>  Find wonderland AND Alice
S=4      <7>   Find rabbit AND S=3
```

finds the records previously located by search result number three that also contain the term 'rabbit'.

### Example 3:

```
S=5      <1>   Find rabbit AND.S watch
```

locates records in which the term 'rabbit' appears in the same **PHrase** field subfield or **Text** field sentence as 'watch', in any order.

### Example 4:

```
S=6      <1>   Find tunnel AND.P well
```

finds all records where 'tunnel' and 'well' exist in the same **Text** field paragraph or **PHrase** field, in any order.

### Example 5:

```
S=7      <2>   Find sister AND.F rabbit
```

locates records in which the terms 'sister' and 'rabbit' are found in the same field.

### Example 6:

```
S=8      <2>   Find book=Alice AND.T chaptnr > 10
```

finds all records where the term 'Alice' and a value greater than ten occur in the same-numbered subfield of a data tuple; that is, the records found are from the book 'Alice's Adventures in Wonderland', Chapters Eleven and Twelve.

### Example 7:

```
S=9      <2>   Find book=Alice AND.N said
```

finds both "Alice said" and "said Alice"..

### Note on example 7:

*The default value for 'n' is 0 (zero), meaning that the operands (words) must be adjacent but can be in any order. Specifying a number other than zero means that there can be up to that number of words between the given search terms.*

## Find – OR operator

### Description

The logical **OR** gives an alternative to one or more search conditions, specifying that at least one of the terms given must occur in the record.

### Syntax

```
Find [search expression] OR term [OR search expression [...]]
```

where *search expression* can be one or more terms to be searched for in the default **Text** and **Phrase** field search area, or in a specific field, field type or view. *Search expression* can also be a record number or a range of record numbers, a reference to one or more previous search results, one or more terms from a **Display** list, a **Fuzzy** search or an indirect search.

### Examples

These examples are taken from the demonstration database **Alice** provided with TRIP.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <17>  Find milk OR butter
```

finds all occurrences of the terms 'milk', 'butter' or both 'milk' and 'butter' within the same record.

#### Example 2:

```
S=3      <20>  Find S=2 OR bread
```

creates a new search result using all records containing the term 'bread' and all records located by search result number two.

#### Example 3:

TRIP executes search orders from left to right, with **AND** or **NOT** combinations carried out before those with **OR** and **XOR**. This can be modified by the use of parentheses. This order:

```
S=4      <8>   Find milk OR bread NOT butter OR cream
```

finds records that contain the term 'milk', *or* those with 'bread' but not 'butter', *or* those with 'cream'. Due to the order of operations in this search request, there is a pair of *implied parentheses* surrounding the words 'bread NOT butter', causing TRIP to treat them as a single phrase.

#### Example 4:

```
S=5      <7>   Find (milk OR bread) NOT (butter OR cream)
```

locates all records containing the terms 'milk' or 'bread', but not 'butter' or 'cream'.

## Find – XOR operator

### Description

The logical **XOR** (eXclusive **OR**) gives an alternative to one or more search conditions, specifying that only one of the terms given may occur in the record.

### Syntax

```
Find [search expression] XOR search expression [XOR search  
expression [...]]
```

where *search expression* can be one or more terms to be searched for in the default **Text** and **Phrase** field search area, or in a specific field, field type or view. *Search expression* can also be a record number or a range of record numbers, a reference to one or more previous search results or terms from a **Display** list, a **FUZZy** search or an indirect search.

### Examples

*Example 1:*

```
S=1      <475> BAsE Alice  
S=2      <3>   Find milk XOR cream
```

finds records that contain *either* 'milk' *or* 'cream', but not both, in the same record.

## Find – NOT operator

### Description

The logical **NOT** excludes the condition to its immediate right, so that any hit record must satisfy the first condition but *not* the second.

Table 13 – List of logical NOT conditions

Operator	Meaning
NOT	'NOT within the same record'
NOT.C	'NOT within the same record component' (for use with record heads and parts)
NOT.E	'NOT within the same record entity' (for use with record heads and parts)
NOT.F	'NOT within the same field'
NOT.P	'NOT within the same TExt paragraph or PHrase field'
NOT.R	'NOT within the same record' (for use with record heads and parts)
NOT.S	'NOT within the same TExt sentence or PHrase subfield'
NOT.T	'NOT from the same subfield number', i.e. within the same data tuple
NOT.W	'NOT within the same word'

### Syntax

```
Find [{search expression|ALL}] [NOT[.x] search expression
NOT[.x] search expression [...]]
```

where *search expression* can be one or more terms to be searched for in the default **T**Ext and **P**Hrase field search area, or in a specific field, field type or view. *Search expression* can also be a record number or a range of record numbers, a reference to one or more previous search results or terms from a **D**isplay list, a **F**UZZy search or an indirect search. 'NOT.x' is the desired **NOT** operator.

The **ALL** keyword is valid only at the very beginning of a **F**ind order when immediately followed by the **NOT** operator. In all other places it is interpreted as the literal "ALL". The **ALL** keyword represents the universal set, i.e. all records in the search set for the currently open database or cluster.

### Examples

Examples one through six are taken from demonstration database **Alice**.

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <2>   Find milk NOT tea
```

finds records containing the term 'milk', but not 'tea'.

*Example 2:*

```
S=3      <1>   Find S=2 NOT thirsty
```

locates all records found by search result number two that do not contain the term 'thirsty'.

*Example 3:*

```
S=4      <56> Find rabbit NOT.S watch
```

finds all records containing 'rabbit', but not 'watch', in a single **PHrase** field subfield or **Text** field sentence and in any order.

*Example 4:*

```
S=5      <8> Find watch NOT.P pocket
```

locates records where 'watch' appears in a **Text** field paragraph or **PHrase** field, but not 'pocket', and in any order.

*Example 5:*

```
S=6      <5> Find TExt=pig NOT.F cat
```

finds records in which the term 'pig' occurs in a **Text** field that does not contain 'cat'.

*Example 6:*

```
S=7      <14> Find person=li# NOT.W lion
```

locates all records containing words beginning with 'li' in **person**, except those containing 'lion'. **NOT.W** is useful when searching databases that contain many compound words, some of which are of no interest to the searcher.

*Example 7:*

```
S=8      <4> Find sname=lindquist AND.T  
         rname=#smith NOT.T day > 19930101
```

finds all records where the sender was 'lindquist', the recipient was 'smith' and the correspondence was dated on or before January 1, 1993 in the same-numbered subfield of a data tuple.

*Example 8:*

```
S=9      <49> Find ALL NOT Alice
```

finds all records that do not contain the term 'Alice' in a **Text** or **PHrase** field.

## Find within fields

### Description

This search type looks for the search term only in the specified field, field type or view.

### Syntax

```
Find {field|fieldtype|view}=term
```

where *field*, *fieldtype* and *view* are the names of the field, field type or view in which to search, and *term* is the phrase, word, character string or value to find.

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <9>   Find speaker=cheshire cat
```

finds the records in which the term 'cheshire cat' exists in the field named **speaker**.

#### Example 2:

```
S=3      <141>      Find person,speaker,txt=white
```

finds the records in which the term 'white' exists in the fields **person**, **speaker**, or **txt**.

#### Example 3:

```
S=4 <49>      Find PERSON <> Alice
```

finds all records where the **person** field does not contain the term 'Alice'.

#### Example 4:

```
S=5      <22> Find PHrase=duchess
```

finds the records in which the term 'duchess' exists in any field of type **PHrase**.

#### Example 5:

```
S=6      <99> BAsE corr
S=7      <0>   DEfine Vlew all_names=rname,sname
S=8      <56> Find all_names=mats
```

opens the demonstration database Corr, defines the Vlew called 'All\_Names' and searches 'All\_Names' for the name 'Mats'.

## Find if Field Content Exists

### Description

TRIP allows searching for the presence or absence of field content using the ["" ] and [#] symbols to designate zero characters (the empty string) and any number of characters (the non-empty string) respectively.

### Syntax

```
Find {field|fieldtype|view}=content
```

where *field*, *fieldtype* and *view* are the names of a field, field type or view to be searched, and *content* is [#] or [\$] to designate content, and ["" ] to signify emptiness.

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <2>   Find TExt=""
```

finds all records in **Alice** in which at least one **TExt** field (in this case, **txt**) is empty.

#### Example 2:

```
S=3      <99> BAsE corr
S=4      <9>   Find trip
S=5      <8>   Find S=4 AND modnote=""
```

finds all records in the last search result performed (S=4) where **Corr**'s **modnote** field is empty.

#### Example 3:

```
S=6      <7>   Find modified=#
```

finds all records in which the field named **modified** is not empty (has content).

#### Example 4:

```
S=7      <475> BAsE Alice
S=8      <473> Find TExt=#
```

locates all records where at least one **TExt** field contains a value.



## Find – Truncation and Masking

### Description

This type of search is used when one or more parts of a search term is not known. The unknown portion may occur anywhere within or between search terms.

TRIP provides three methods of blocking unknown words or word sections while searching: *word masking*, *word truncation* and *character masking*.

#### Word Masking

Word masking allows TRIP to ignore one or more words that may occur between any two search terms. Both *proximity operators* and the *logical AND* can be used in word masking, the difference being that term order is important in proximity searching.

For example, in the sentence ‘ “Well, a ‘Rath’ is a sort of green pig: but ‘Mome’ I’m not certain about.” ’ in the demonstration database **Alice**, the search order

```
Find rath $ $ $ $ $ $ $ mome
```

substitutes one dollar symbol [\$] for each word that separates the search terms ‘rath’ and ‘mome’.

Refer to the sections entitled ‘Find–Proximity Searching’ and ‘Find–Logical AND’ in this manual for more information regarding word masking.

#### Word Truncation

Word truncation ‘abbreviates’ a word by one or more characters by concealing these prefix or suffix characters from the search. For example, to find all the terms beginning with ‘ja’ in **Alice**, you could use the order

```
Find ja#
```

Substituting the pound symbol [#] for all characters after the ‘a’ causes all such characters to be accepted in a search, making this request locate terms such as ‘jar’, ‘Jack’, ‘jam’, ‘Jabberwock’ and ‘Jabberwocky’.

To find all the words ending in ‘tation’, the order

```
Find #tation
```

would ignore all characters preceding the first ‘t’, and might locate records containing ‘station’, ‘consultation’, ‘invitation’ and ‘irritation’.

#### Character Masking

Character masking blocks one or more characters within a word. For example, the order

```
Find pa#y
```

causes all characters between the ‘a’ and ‘y’ to be accepted, locating terms like ‘particularly’, ‘party’, ‘passionately’, ‘patiently’ and ‘pay’.

Any number of masking symbols may be used in any one term, and terms containing truncation or masking symbols may be used in a search order that contains any of the other aspects of searching on textual contents.

Conditions which relate to truncated and masked terms can be mixed in search orders with those that relate to full term content searching.

The usual system default symbols are defined as follows:

**Table 14 – Default system search masking symbols**

Character	Function
#	indicates any character string of zero, one or more characters
!	designates exactly one unknown character
:	signifies one character or none

In addition to the masking symbols previously discussed, the dollar symbol [\$] is also used to locate different forms or spellings of a word, compound words and varieties of names, as follows (the [ ] symbol signifies a single space between characters):

**Table 15 – User of the \$ symbol in search masking**

Symbol	Function
\$	at the beginning or end of a term, denotes any character string of any length, including zero characters
\$	when within a term, denotes exactly one unknown character
\$\$	when within a term, denotes exactly two unknown characters
\$\$\$	when within a term, denotes any character string of any length, including zero characters

## Examples

### Example 1:

```
S=1      <99>  BAsE corr
S=2      <9>   Find ab:l#
```

finds 'able' and 'ability', but not 'table' or 'stability'.

Examples two through six use the demonstration database Alice.

### Example 2:

```
S=1      <475> BAsE Alice
S=2      <28>  Find !ose
```

finds 'lose', 'nose' and 'rose', but not 'close', 'goose' or 'suppose'.

### Example 3:

```
S=3      <20>  Find advi:e
```

finds 'advice' and 'advise'.

### Example 4:

```
S=4      <81>  Find $ouse$
```

finds 'curiouser', 'dormouse', 'house', 'housemaid', 'houses' and 'mouse', in any field of type **PHrase** or **TExt**.

**Example 5:**

```
S=5      <5>  Find txt=a$t
```

finds 'act', 'ant' and 'art', but not 'active', 'constant', 'darted', 'at' or 'aloft' in **Alice's txt** field.

**Example 6:**

```
S=6      <77> Find TExt=g$$d
```

finds 'glad', 'gold' and 'good', but *not* 'god' or 'gonad'.

**Example 7:**

```
S=1      <99> BAsE corr
```

```
S=2      <9>  Find content=acc$$$e
```

finds 'acceptable', 'accessible', 'accordance' and 'accurate' in the field named **content**.



## Find – Proximity Searching

### Description

Proximity searching allows a searcher to locate records containing only search terms occurring within a specified distance of one another in the text, for example, within the same sentence or subfield, or separated from one another by a certain number of words.

The essential difference between proximity searching and the logical **AND** is that in proximity searching the *term order* is important. The symbol **AND.S**, meaning ‘**AND** in the same sentence or subfield’ is quite accurate; however the proximity operators must be used to obtain precision greater than this.

The simplest proximity operator is the **SPace** character, meaning ‘adjacent to and in the order specified’.

The other proximity operators specify the number of terms which are not being searched for, which are positioned between the terms being searched for. These operators are (the [ ] symbol represents a single space):

Table 16 – Proximity search operator usage

Symbol	Meaning
[ ]	denotes zero terms or one term
[ ] ... [ ]	denotes zero, one or more terms
[ ] [ ] [ ]	denotes zero to two terms
[ ] [ ] [ ] [ ]	denotes zero to three terms
[ ] [ ] [ ] ... [ ] [ ] [ ]	any number of stops separated by spaces indicating any number of terms, the ellipsis [...] signifying ‘an infinite number of’
[ ] \$ [ ]	when outside a word, denotes exactly one term
[ ] \$ [ ] \$ [ ]	when having a single space between symbols, indicates exactly two words between search terms
[ ] \$ [ ] \$ [ ] \$ [ ]	when having a single space between symbols, indicates exactly three words between search terms
&	when followed by a term being searched for, specifies that the character string immediately following the & should be found only when it appears at the beginning of a subfield or sentence

As these operators enable further refinement of searches within a sentence, the search terms for all the examples of this section are understood to coexist within the same sentence.

Conditions which relate to proximity searches can be mixed in search orders with those that relate to full and truncated or masked term content searching.

### Examples

The [ ] symbol represents a single space in the examples that follow.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <5>   Find speaker=frog  footman
```

finds the records in which these terms exist next to each other and in the order specified, in the field named **speaker**.

#### Example 2:

```
S=1      <475> BAsE Alice
S=2      <1>   Find TExt=the  frog  footman
```

finds the records in which the terms 'the', 'frog' and 'footman' exist beside one another and in the order specified in any **TExt** field.

If the field name or type is omitted, searching occurs in all **TExt** and **PHrase** fields by default.

#### Example 3:

```
S=1      <475> BAsE Alice
S=2      <6>   Find frog  footman
```

searches all **PHrase** and **TExt** fields for records in which the terms 'frog' and 'footman' exist next to each other and in the order specified.

#### Example 4:

```
S=1      <99>  BAsE corr
S=2      <1>   Find stockholm [...] november
```

finds records which contain the terms 'Stockholm' and 'November' in the same sentence or subfield, with any number of terms between them and in the order specified. There are no spaces between the three [...] characters here.

#### Example 5:

```
S=1      <99>  BAsE corr
S=2      <14>  Find PHrase=p o box [...] #1#
```

finds records which contain the terms 'P O Box' and the value '1' within any number of terms of each other, in the same subfield of any field of type **PHrase**, and in the order specified. There are no spaces between the three [...] characters.

#### Example 6:

```
S=1      <475> BAsE Alice
S=2      <1>   Find rabbit [...] watch
```

finds records which contain the two terms 'rabbit' and 'watch' separated by no more than three terms, and in the order specified. Each [...] character is separated by a space in this example.

*Example 7:*

```
S=1      <475> BAsE Alice
S=2      <1>   Find latitude _$$_$ longitude
```

finds records which contain the two terms 'latitude' and 'longitude' separated by exactly two terms, and in the order specified. A single space separates the two [\$] characters here.

*Example 8:*

```
S=1      <99> BAsE corr
S=2      <9>   Find PHrase=&mats l:fstr:m
```

finds records which contain the two terms 'Mats' and 'Löfström' as the very first terms in any **PHrase** field subfield, in the order specified. This order will *not* locate records containing 'Mr. Mats Löfström'.



## Find – Previous Searches

### Description

The results of a previous search or searches can be combined with either the results of a current search or other previous searches. Any number of previous search results can be combined in this manner using any of the operators, as long as they are from the same database(s) and the database(s) is/are currently open.

There are four ways to specify previous searches: one or more search result number(s), 'TO a number', 'a number TO a number' and 'FRom a number', separated by commas.

The search result number(s) and the interval 'number TO a number' may be repeated in the search order with different parameters, but 'TO a number' and 'FRom a number' can only be used once per order.

In any order that uses a mixture of **TO** and **FRom** intervals, the interval 'TO a number' must always be given first, then 'a number TO a number', and finally 'FRom a number'. Were all three examples to be combined in the same order, they would have to appear in the order given above, e.g.:

```
Find S=TO 10, 17 TO 23, FRom 30
```

The condition **S=0** always refers to the most recent search result. If the **S=** condition in a search order is omitted, the default search result **S=0** is assumed.

Previous searches can be combined with both full term and truncated or masked content searching.

### Syntax

For single values:

```
Find S=n[,n[,...]]
```

For 'TO a number' intervals:

```
Find S=TO n
```

For 'FRom a number' intervals:

```
Find S=FRom n
```

For 'number TO a number' intervals:

```
Find S=n TO n[,n TO n[,...]]
```

where **S** represents 'search result' and *n* is a search result number.

### Examples

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <5>   Find red king
S=3      <4>   Find red knight
S=4      <5>   Find S=2
```

re-finds the records previously located by search result number two.

**Example 2:**

```

S=1      <475> BAsE Alice
S=2      <5>   Find red king
S=3      <51>  Find red queen
S=4      <1>   Find and S=2

```

finds those records common to the most recent search (S=3) and search result number two.

**Example 3:**

```

S=1      <475> BAsE Alice
S=2      <29>  Find PHrase=white king
S=3      <53>  Find PHrase=white queen
S=4      <27>  Find PHrase=white knight
S=5      <34>  Find PHrase=white rabbit
S=6      <2>   Find PHrase=gentleman in white
S=7      <99>  Find S=2 TO 4,0

```

collects all records previously located by searches two, three, four, and the most recent search (S=6) into a single search result (note that no explicit Boolean operator is present).

**Example 4:**

```

S=1      <475> BAsE Alice
S=2      <29>  Find PHrase=white king
S=3      <53>  Find PHrase=white queen
S=4      <27>  Find PHrase=white knight
S=5      <34>  Find PHrase=white rabbit
S=6      <2>   Find PHrase=gentleman in white
S=7      <99>  Find S=2 TO 4,0
DELeTe S=1,7
S=8      <73>  Find S=TO 3, FRom 6

```

After the deletion of search result numbers one and seven, gathers all records previously located by all searches into a single search set, with the exception of search results four and five.



## Find Terms From a Display List

### Description

A search order using this type of condition is used to combine terms from a display list.

A single term number or list or range of term numbers may be used to select terms from a **Display** list to form a **Find** order. Multiple terms are delimited by commas. When using lists and/or ranges, intervals are formed using **TO** and **FRom**, e.g. 'TO 3', '5 TO 7' and 'FRom 10'.

There are three types of intervals: 'TO a number', 'a number TO a number' and 'FRom a number'. In any order that uses a mixture of **TO** and **FRom** intervals, the interval 'TO a number' must always be given first, then 'a number TO a number', and finally 'FRom a number'. Were all three examples to be combined in the same order, they would have to appear in the order given above.

Display list term numbers can be mixed in search orders with those relating to full term and truncated or masked content searching, as well as the results of previous searches.

### Syntax

For single values:

```
Find T=n[,n[,...]]
```

For 'TO a number' intervals:

```
Find T=TO n
```

For 'FRom a number' intervals:

```
Find T=FRom n
```

For 'number TO a number' intervals:

```
Find T=n TO n[,n TO n[,...]]
```

where **T** represents 'term' and *n* is a term number.

### Examples

The following examples are intended for use with the demonstration database **Alice**.

#### Example 1:

```
S=1      <475> BAsE Alice
Display #ous#
S=2      <5>   Find T=3
```

finds all records containing 'cautiously', the third term given in the display list.

#### Example 2:

```
S=3      <26> Find T=5 TO 7
```

finds those records which contain 'curious', 'curiouser' or 'curiously', the terms numbered five, six and seven in the current display list.

#### Example 3:

```
S=4      <37> Find T=TO 4, 12, FRom 31
```

finds those records which contain the terms 'anxious', 'anxiously', 'cautiously', 'contemptuously', 'furiously' or 'vicious', which are numbered one through four, twelve and from thirty-one upwards in the current display list.



## Find – Order of Operator Precedence

### Description

TRIP executes search orders from left to right, with the exception that a combination with **OR** is executed after combinations with **AND** or **NOT**. However, as in mathematics, the order can be altered using parentheses.

Parentheses must be balanced, and can be nested. If the order of a complex search order is unclear, parentheses may be used to clarify how TRIP interprets the order of the search conditions.

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <13> Find bread AND butter OR sugar
```

finds records which contain both 'bread' and 'butter' or 'sugar'.

#### Example 2:

```
S=1      <475> BAsE Alice
S=2      <13> Find bread AND butter OR sugar
S=3      <11> Find bread AND (butter OR sugar)
```

finds records which contain the term 'bread' *and* either 'butter' or 'sugar'.

#### Example 3:

```
S=1      <99> BAsE corr
S=2      <2> Find(sname=mats AND lin#)NOT(sname=mats g AND
lin#)
```

locates all records containing the name 'mats' and terms beginning with 'lin', but not records containing 'mats g' and terms beginning with 'lin'.

## Find – FREquency

### Description

Search for terms based on occurrence counts by using the FREquency attribute.

### Syntax

```
Find field=term FREquency operator n
```

where *field* is the name of a database field, *term* is the search term to be located, *operator* is any of the permissible operators (= >= <= > < or <>) and 'n' is the FREquency value to be used.

*NB: This is guaranteed to work correctly only when searching single databases.*

### Examples

#### Example 1

```
S=5      <7>      Find person=a# FREquency < 10
```

finds the records where any words in the person field beginning with the letter 'a' occur less than ten times

## Find FUZZ

### Description

Notes:

- The FUZZ() modifier to the Find command is not to be confused with the CCL command, 'FUZZ'
- For more help on fuzzy searching, refer to Chapter Five, 'Advanced Features', of the TRIPmanager User Guide

Fuzzy searching implies searching for similarity, a capability that can prove quite useful when looking for differently spelled or frequently misspelled terms and using **Find FUZZ()** returns records containing the search term, as well as terms that are considered similar to the search term.

In addition to the search term, the **FUZZ()** modifier can take four optional comma separated parameters that alter the performance of the fuzzy search.

### Parameters

1. 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

2. The second parameter is depreciated and has no effect in normal fuzz searching and can be left out; however, the leading comma must be used if any following parameters are entered.
3. The third parameter sets the number of top ranked search candidates to include with a **Find FUZZ()**.

Default Value: 2

Note:

*See the **Display FUZZ()** command for details on how to obtain a list of search candidates used in a **Find FUZZ()** search.*

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

Default=1

Note:

*The default is the recommended search algorithm.*

Refer to the section '**DEfine FUZZ**' in this manual for more information on redefining the default values of fuzzy parameters, and '**Display FUZZ()**' for help with listing fuzzy terms.

## Syntax

```
Find FUZZ (term[,percentage][,depreciated][,find-terms]
           [,algorithm])
```

where *term* is the search expression to find, *percentage* is the desired percentage of word matching (range: 1\*100); *depreciated* and *find-terms* have no effect; *algorithm* is the search algorithm to be used (1 or 2).

## Examples

The following examples are for use with the demonstration database **Alice**.

### Example 1:

```
S=1      <475> BAsE Alice
S=2      <12> Find FUZZ(late)
```

uses the default **Find FUZZ()** settings to return all records in the demonstration database **Alice** having a 75% similarity to 'late' and contain the two top ranked search candidates (i.e. 'late' itself and 'later') with the 'n-grams' algorithm.

### Example 2:

```
S=3      <40> Find FUZZ(late,,,7)
```

sets the number of top ranked search candidates to 7 and so uses the 'n-grams' algorithm to return records containing the first seven terms that have a 75% similarity to the search term; i.e. 'late' itself, plus 'later', 'rate', 'slate', 'gate', 'hate' and 'plate'.

### Example 3:

```
S=4      <19> Find FUZZ(late,84,,4)
Show SORT=Frequency
```

uses a percentage similarity of 84%, the 'n-grams' algorithm and the four top ranked search candidates, to return records containing 'late' itself, plus 'later', 'slate' and 'plate'.

## Find – Relational Operators

### Description

Relational operator searching may be applied to fields with text or numeric content.

The 'equal to', 'greater than', 'less than', 'greater than or equal to' and 'less than or equal to' relational operators can be applied to either text or numeric fields. There is also a text interval symbol for use in **Text** and **Phrase** fields. For **Number**, **Integer**, **Date** and **Time** fields where the field name rather than a field type is specified, the numeric intervals 'From a number', 'TO a number' and 'number TO a number' and 'not equal to' allow the specification of exact values and ranges of values.

The relational operators are:

Table 17 – Relational operators for the Find order

Relational Operator	Meaning
=	'equal to the following value'
<	'less than the following value'
>	'greater than the following value'
<=	'less than or equal to the following value'
>=	'greater than or equal to the following value'
[ ]	text interval—for use with <b>Text</b> and <b>Phrase</b> fields only
< >	'not equal to the following value'
=From	'from and inclusive of the value that follows'
=TO	'up to and inclusive of the following value'
TO	used between two values to denote 'from and inclusive of the preceding value, and to and inclusive of the following value'

The [ ] symbol represents a single space.

**From** (shortest form: **FR**) is used with lists of record, term or search result numbers separated by commas to specify an interval or range. This interval consists of **From** and a numerical value, and must be placed last in a list of numbers and intervals.

Values used in searches for the numerical field types **Number**, **Integer**, **Date** and **Time** must be in the correct format.

Combinations of exact values and ranges may be entered against a single field name. Multiple values and ranges are separated by commas.

Conditions which relate to textual content searching can be mixed in search orders with those that relate to any numerical content searching.

## Syntax

For alphanumeric relational operations:

```
Find field{=|<|>|<=|>=|<>}term
```

For text relational operations:

```
Find field=term .. term
```

For numeric relational operations:

```
Find field=n[,n[,...]]
```

```
Find field=TO n
```

```
Find field=FRom n
```

```
Find field=n TO n[,n TO n[,...]]
```

where *field* is the name of the field to be searched, *term* is the term against which each potential hit term will be measured and *n* is a value.

## Examples

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <4>   Find rname=arturo martinez
```

finds all records containing the phrase 'Arturo Martinez' in the **PHrase** field **rname**.

*Example 2:*

```
S=1      <475> BAsE Alice
S=2      <43>  Find chaptnr=3
```

finds all records where **chaptnr** is equal to three.

*Example 3:*

```
S=1      <99>  BAsE corr
S=2      <18>  Find rname >= mats and rname < mr
```

finds all records where the content of at least one subfield of **rname** has an alphanumeric value greater than or equal to 'mats' and less than 'mr'. This example finds all occurrences of 'Mats Lindquist', 'Mats G. Lindquist' and 'Mats Löfström'.

*Example 4:*

```
S=1      <475> BAsE Alice
S=2      <394> Find chaptnr > 2
```

finds all records where **chaptnr** is greater than two.

*Example 5:*

```
S=1      <475> BAsE Alice
S=2      <261> Find chaptnr <= 6
```

finds all records where **chaptnr** is less than or equal to six in database **Alice**.

*Example 6:*

Upper and lower limits of text intervals are written as full stops between search terms.

```
S=1      <99>  BAsE corr
S=2      <8>   Find rname=dr .. dz
```



finds all records where the alphanumeric value of any item in **rname** is greater than or equal to 'dr' and less than 'dz'. In this case, the order locates all names preceded by the title of 'Dr'. The [ ] symbol above represents a single space.

*Example 7:*

```
S=1      <475> BAsE Alice
S=2      <417> Find chaptnr <> 9
```

finds all records where **chaptnr** is not equal to nine.

*Example 8:*

```
S=1      <24> BAsE carroll
S=2      <8> Find chaptnr=TO 4
```

finds all records where **chaptnr** is less than or equal to four.

*Example 9:*

```
S=1      <24> BAsE carroll
S=2      <8> Find chaptnr=FRom 9
```

finds all records where **chaptnr** is greater than or equal to nine in database **Carroll**.

*Example 10:*

```
S=1      <24> BAsE carroll
S=2      <6> Find chaptnr=10 TO 12
```

finds all records in **Carroll** where the value of **chaptnr** is ten to twelve inclusive.

*Example 11:*

```
S=1      <24> BAsE carroll
S=2      <14> Find chaptnr=TO 3, 6 TO 7, FRom 11
```

finds all records in **Carroll** except those where the value of **chaptnr** is four, five, eight, nine or ten.

*Example 12:*

```
S=1      <24> BAsE carroll
S=2      <3> Find watch and chaptnr=1,3,FRom 11
```

locates all records containing the word 'watch' where the value of **chaptnr** is one, three or eleven and up in database **Carroll**.

*Example 13:*

```
S=1      <24> BAsE carroll
Display #use
S=2      <21> Find T=TO 4,6,FRom 8
```

creates a search result from a display list in **Carroll**, using term numbers one through four, six, and those numbered eight and up ('applause', 'because', 'cause', 'dormouse', 'house', 'pause' and 'use') in a search order.

*Example 14:*

Dates may be given in full or as the year only, or as year and month only. The field **moddate** in demonstration database **Corr** is of type **DAt**e, where

```
S=1      <99> BAsE corr
S=2      <7> Find moddate=199306
```

finds all records modified in June of 1993 (the default date format is YYYY-MM-DD).

*Example 15:*

```
S=1      <99>  BAsE corr  
S=2      <2>   Find moddate=TO 19930605
```

finds all records in **Corr** where **moddate** contains a value up to and including June 5, 1993.



## Find Record number

### Description

Each record in a TRIP database has a unique *record number*. These numbers are allocated incrementally as the records are added to the database and may not be reassigned, even if a record is deleted and its number is freed. This allows successful direct searching on record numbers whether or not the database has been indexed since records were last added.

There are four ways to specify record numbers: one or more record number(s), 'TO a number', 'a number TO a number' and 'FRom a number', separated by commas.

In any order that uses a mixture of **TO** and **FRom** intervals, the interval 'TO a number' must always be given first, then 'a number TO a number', and finally 'FRom a number'. Were all three examples to be combined in the same order, they would have to appear in the order given above.

The search result number(s) and the interval 'number TO a number' may be repeated in the search order, but 'TO a number' and 'FRom a number' can only be used once apiece.

Record number searches can be combined with both full term and truncated or masked content searching.

### Syntax

For single values:

```
Find R=n[,n[,...]]
```

For 'TO a number' intervals:

```
Find R=TO n
```

For 'FRom a number' intervals:

```
Find R=FRom n
```

For 'number TO a number' intervals:

```
Find R=n TO n[,n TO n[,...]]
```

where **R** represents 'record' and *n* is a record number.

### Examples

The following examples are intended for use with demonstration database **Alice**, and should be worked consecutively.

*Example 1:*

```
S=1      <475> BAsE Alice
```

```
S=2      <1>   Find R=1
```

finds the record with record number one. If this record has been deleted, the search returns no hits.

*Example 2:*

```
S=3      <475> Find R=FRom 1
```

finds all the records in the database.

*Example 3:*

```
S=4      <100> Find R=TO 100
```

finds the records with record numbers less than or equal to one hundred.

*Example 4:*

```
S=5      <11> Find R=10 TO 20
```

finds records with record numbers greater than or equal to ten, yet less than or equal to twenty.



## Find SAVE

### Description

Finds and runs the saved procedure included in the command argument.

Provided that such a procedure contains only search related commands (e.g. **FIND**, **SAVE**, **DEFINE**), then TRIPsystem will locate and automatically run the procedure if the command **FIND SAVE *myproc*** is used.

### Syntax

```
Find SAVE myproc
```

where ***myproc*** represents name of the previously saved procedure.

### Examples

Example 1:

```
S=1    <475>    BAsE Alice
S=2    <426>    Find Alice
S=3    <46>     Find S=6 AND rabbit
S=4    <1>      Find S=7 AND mushroom
SAVE PROC1
FIND SAVE PROC1
```

Results in the saved procedure named ***PROC1***, being located and automatically run, with the resultant output added to the search history as follows:

```
S=5    <475>    BAsE Alice
S=6    <426>    Find Alice
S=7    <46>     Find S=6 AND rabbit
S=8    <1>      Find S=7 AND mushroom
```

## Find TStamp

### Description

Each record in a TRIP database has the date and time of creation or last modification stored in an area called the *timestamp*. This area can be searched using all the standard numerical date and time searching techniques.

In a timestamp search order a time cannot be entered without a date. Further, if a time is included in the expression, the greater-than and less-than symbols [**<**] and [**>**] may *not* be used.

### Syntax

To search using single values:

```
Find TStamp=date [time][,date[time][,...]]
```

To search using 'TO a number' intervals:

```
Find TStamp=TO date[time]
```

To search using 'FROM a number' intervals:

```
Find TStamp=FRom date[time]
```

To search using 'a number TO a number' intervals:

```
Find TStamp=date[time] TO date[time][,date[time] TO  
date[time][,...]]
```

To search using less-than and greater-than symbols:

```
Find TStamp{<|>|<=|>=}date
```

where *date* and *time* are date and time values respectively.

### Examples

These examples are for use with demonstration database **Corr**, and should be worked consecutively. The default date format is YYYY-MM-DD. Further examples may be found in the 'Timestamp' section of Chapter Five, 'Advanced Features', in the *TRIPclassic User Guide*.

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <7>   Find TStamp=1993
```

finds all records containing the year '1993' in their timestamps.

*Example 2:*

```
S=3      <3>   Find TStamp=FRom 19930618 11:52:12
```

locates all timestamps containing the date intervals and times given above.

*Example 3:*

```
S=4      <7>   Find TStamp=19930101 00:00:00 TO  
19940401 23:59:59
```

locates all records created or modified between January 1, 1993 and April 1, 1994 inclusive.

## Find TODAY

### Description

The TODAY() function can be used instead of a DATE literal. By default, it evaluates to the current date. A date offset can be specified by day, week, month or year to instead have the function to evaluate to an earlier or later date.

### Syntax

The general syntax is:

```
Find DATEFIELD=Today([-] number offset)
```

Where the *offset* specification is a string of max 3 characters in length. This string specifies the what given *number* represents and how the date shall be calculated. Its syntax is:

```
(Y|M|D|W) [ (Y|M|D|W) [ (Y|M|D) ] ]
```

The **first position** of the offset specification indicates offset period. This position is mandatory and must have one of the following values:

D	Number of days to add or subtract from the current date.
W	Number of weeks to add or subtract from the current date. The calculated date will be set to the first day of the week unless the second position argument is set to <b>D</b> , in which case the same week day in the resulting week will be used.
M	Number of months to add or subtract from the current date. The calculated date will be set to the first day of the month unless the second position is set to <b>D</b> .
Y	Number of years to add or subtract from the current date. The calculated date will be set to the first day of the year unless the second position is set to <b>D</b> or <b>M</b> .

The **second position**, if provided, should normally have the same value as the first. It determines if the resulting date will be the first date in the offset period (first day of week, month or year), or one of the valid alternatives. Valid values are:

D	<p>If the first position is <b>W</b>(eek), the calculated date will be set to the same day of week as is current.</p> <p>If the first position is <b>M</b>(onth), the calculated date will be set to the same day of the month as the current date. If the resulting date is in a shorter month and the current day of month does not exist in the target month, the date will be set to the last day of the target month instead.</p> <p>If the first position is <b>Y</b>(ear), the calculated date will be set to the same month and day of the month as the current date.</p> <p>No-op if the first position is D(ay).</p>
W	The calculated date will be set to the first day of the week. Only valid if the first position also is <b>W</b> (eek).
M	The calculated day will be set to the first day of the month.

	<p>If the first position is <b>Y</b>(ear), the calculated date will be set to the first day of the month.</p> <p>Not valid if first position is <b>W</b>(eek) or <b>D</b>(ay). No-op if first position is also <b>M</b>(onth).</p>
Y	The calculated date will be the first day of the year. Only valid if the first position also is Y(ear)

The **third position**, if provided, specifies how many parts of the date will be included in the resulting DATE literal. The default is D(ay), which means that the full date (year, month and day) is returned. Valid values are:

D	All parts of the date literal are present. This is the default.
M	The date literal includes year and month only.
Y	The date literal only includes the year.

## Examples

These examples are for use with demonstration database **Corr**. The default date format is YYYY-MM-DD.

*Example 1: With default behaviour, corresponding to DEFINE TODAY NO EXPAND:*

```
S=1      <99>  BAsE CORR
S=2      <95>  Find DAY=FRom TOday (-42Y)
S=3      <90>  Find DAY=FRom TOday (-42YD)
S=4      <7>   Find DAY=TOday (-42YYY)
```

*Example 2: With DEFINE TODAY EXPAND declared, run on October 23<sup>rd</sup> 2023 using TODAY(-42Y)*

```
S=5      <95>  Find DAY=FRom 1981-01-01
```

*Example 3: With DEFINE TODAY EXPAND declared, run on October 23<sup>rd</sup> 2023 using TODAY(-42YD)*

```
S=6      <90>  Find DAY=FRom 1981-10-23
```

*Example 4: With DEFINE TODAY EXPAND declared, run in year 2023 using TODAY(-42YYY)*

```
S=7      <7>   Find DAY=1981
```



## Find DUPLICATE

### Description

The modifier **DUPLICATE** can be used to find “identical” records in a database. A list of fields is specified to be checked, and TRIP will return a search set depending on the type of duplicate search selected.

When searching for duplicates in the db, the record with the lowest record number will become the “original” and additional records with the same contents in the specified fields will then be “duplicates”.

### Syntax

```
Find DUPLICATE(fieldname1, fieldname2,...,n])
```

Where ‘*fieldname1, fieldname2, ...*’ is a comma separated list of field names and ‘*n*’ is an optional value of 0 (or no value – find all duplicates), 1 (find only original records having duplicates, or 2 (both 0 and 1 together).

### Examples

The following examples all use the demonstration database ‘corr’:

#### Example 1

All “duplicates”:

```
S=2 <17> Find DUPlicate(rname, sname, saddr, raddr)
S=3 <17> Find DUPlicate(rname, sname, saddr, raddr, 0)
```

#### Example 2

The “original” records, i.e. those records that have duplicates:

```
S=4 <9> Find DUPlicate(rname, sname, saddr, raddr, 1)
```

#### Example 3

Both of the above together:

```
S=5 <26> Find DUPlicate(rname, sname, saddr, raddr, 2)
```

**Note:**

*Only the content of the first subfield of every field is used when comparing fields to find duplicates.*

## Find User

### Description

This function can be useful when setting up read/write scopes for a database if there are restrictions on access to certain records based on the TRIP user name.

If LDAP authentication is used, the LDAP user name may differ from the TRIP user name. To differentiate between the two, this function takes an optional argument to indicate which user name it shall resolve to; the external (LDAP) user, or the internal (TRIP) user. The default, if no argument is specified, is to match the internal user.

### Syntax

To search for the name of the user currently logged in:

```
Find User([INTERNAL|EXTERNAL])
```

### Examples

The following examples all use the Connector database design type, as available with TRIP 8.1 and later.

#### *Example 1:*

Match internal user:

```
Find I_TACL_USERS=USER()
```

#### *Example 2*

Match external user:

```
Find I_ACL_USERS=USER(EXTERNAL)
```

## Find GROUP

### Description

This function can be useful when setting up read/write scopes for a database if there are restrictions on access to certain records based on the names of the TRIP groups that the currently logged in user is a member of.

If LDAP authentication is used, the the names of the groups the LDAP user is a member of are also available. To differentiate between the two sets of groups, this function takes an optional argument to indicate which user name it shall resolve to; the external (LDAP) user, or the internal (TRIP) user. The default, if no argument is specified, is to match the internal user.

### Syntax

To search for the names the groups that the currently logged in user is a member of:

```
Find GROUP ([INTERNAL|EXTERNAL])
```

### Examples

The following examples all use the Connector database design type, as available with TRIP 8.1 and later.

#### *Example 1:*

Match at least one of the internal groups the TRIP user is a member of:

```
Find I_TACL_GROUPS=GROUP()
```

#### *Example 2*

Match at least one of the external groups that the LDAP user is a member of:

```
Find I_ACL_GROUPS=GROUP(EXTERNAL)
```

## Find THESaurus

### Description

The **CT**, **BT**, **NT** and **RT** elements in a thesaurus record can be used in **Find** orders, to look up terms in the thesaurus and search for them in all of the specified fields of the target database(s).

A thesaurus must be defined using **DEfine THESaurus** prior to searching.

Refer to the 'Display–Thesaurus Modifiers' section of this manual and Chapter Six, 'Thesaurus Searching' of the *TRIPclassic User Guide* for further information on the thesaurus.

### Syntax

```
Find {CT|BT|NT|RT|UP|DOWN} (term)
```

where *term* is the term with which the thesaurus **CT**, **BT** etc. will be searched.

### Examples

The following examples use demonstration database **Alice** and its thesaurus, **Thesali**.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <0>   DEfine THESaurus=thesali
S=3      <4>   Find CT($fly)
```

defines the reference thesaurus as **Thesali**, looks up terms ending with 'fly' in the **CT** or **UF** fields in **Thesali**, picks the **CT** terms of those records and searches the **PHrase** fields of the open database (**Alice**) for those terms.

#### Example 2:

```
S=4      <6>   Find NT(footmen)
```

finds records containing the narrower terms of 'footmen' ('fish footman' and 'frog footman') in any **Alice PHrase** field.

#### Example 3:

```
S=5      <16>  Find UP(king's messenger)
```

finds records containing 'king's messenger' or one of its broader terms. **DOWN** is used in a way parallel to **UP**.

#### Example 4:

```
S=6      <4>   Find DOWN(food)
```

locates all the terms below or downwards from 'food', in this case 'mutton' and 'pudding'.

#### Example 5:

```
S=7      <1>   Find BT('footmen')
```

locates the thesaurus records with the word 'footmen' in the **BT** field, with exact matching of the term or phrase surrounded by single quotes, and uses these terms to search the target database (in this case finding 'Alice's servants' in **Alice's person** field).

*Example 6:*

```
S=8      <9>   Find TExt=NT(food)
```

searches for 'food' and its narrower terms (finding, in this case, 'mutton' and 'pudding') in all **TExt** fields. The target specification of a **Define** order may be temporarily overridden by including a field name or type in the search order.



## Find – Mapped Transaction Sets

### Description

Indirect search orders take the vocabulary of one field from a set of records in one database (referred to as the *source*), and apply these as search terms to another database (referred to as the *target*).

The essential element of an indirect search is the *virtual field*, which is defined by a **DEfine MAP** order. This order defines the source database; the source field (which must be of type **PHrase**), the field or field type in the source database to which the *argument* is applied, and, in the indirect search order, the field or type of field to which the final search is applied in the target.

Refer to the section on **DEfine MAP** in this guide for further information.

### Syntax

For general searching:

```
Find [{field|fieldtype|view}=] virtualfield(argument)
```

For searching using a search result:

```
Find S=n.virtualfield
```

where:

1. *field*, *fieldtype* and *view* are fields, field types and views in which to search,
2. *n* is a search result number,
3. *virtualfield* is a temporary or 'ghost' field created by the user to reference the source database phrase field, and does not truly exist as part of any database definition,
4. *argument* can include single terms, truncated terms or a combination of these separated by proximity and/or logical operators.
5. the operation of all of these aspects is the same as for direct searching.

### Examples

The following examples use the demonstration databases **Carroll** and **Alice** provided with TRIP, and assume that the **DEfine MAP** order shown (which maps **Person** from **Alice** into **Carroll**) is being used for all examples. All examples should be performed consecutively.

*Example 1:*

```
S=1      <499> BAsE two=carroll,alice
S=2      <0>   DEfine MAP people=alice.person
S=3      <189> Find people(turtle)
```

searches **Alice** for the term 'turtle' in **TEText** and **PHrase** fields, extracts the terms from the field **person** of the hit records and searches the cluster database **Two** for these terms in the **TEText** and **PHrase** fields.

*Example 2:*

```
S=4      <4>   Find verse=people(turtle)
```

as above, but the search for the extracted terms is in the field **verse** of cluster database **Two** rather than all **Text** and **PHrase** fields.

*Example 3:*

```
S=5      <18> Find speaker=people(turtle) AND
           person=march hare
```

as above, but the search for the extracted terms is in the field **speaker** of **Two**, and combines the result of a direct search for the term 'march hare' in the **person** field.

*Example 4:*

```
S=6      <49> Find people((knave OR queen#) AND.C heart#)
```

as above, only the first search in **Alice** is slightly more complex and contains truncation and logical operators.

*Example 5:*

```
S=7      <275> Find people(knave . hearts OR white rabbit)
```

as above, only the first search in **Alice** is slightly more complex and contains proximity operators.

*Example 6:*

To refer to the results of a previous search, use the **S=n** qualifier:

```
Display S=3.people
```

reproduces a display list of the extracted terms.

*Example 7:*

```
S=8      <403> Find S=3.people
```

finds the records which contain those terms. No argument is required here.

## Find – Internal Transaction Sets

### Description

This kind of indirect search order applies the vocabulary of one **Non-TEXT** field from a previous search result to the open database as search terms. The condition can be used in the same way as any other condition in a search order.

The records extracted from one field from one search result set form the *transaction set*, which is defined by the combination of the search result number and the field.

No **Define MAP** order is required in order to use internal transaction sets. All the information required for the **Find** order is contained within it.

### Syntax

```
Find [{field|fieldtype|view}=] n.sourcefield
```

where:

- *field*, *fieldtype* and *view* are fields, field types and views in which to search,
- *n* is a search result number,
- and *sourcefield* is a **Non-TEXT** field that will supply the search terms to be used.

### Examples

These examples have been taken from the **Corr** demonstration database provided with TRIP.

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <5>   Find telex
S=3      <1>   Find memo
S=4      <27>  Find letter
S=5      <2>   Find 3.rname
```

finds all records that contain any one of the names in the **rname** field of the third search result.

*Example 2:*

```
S=6      <15>  Find rname=0.sname
```

extracts the terms in **sname** in the records found by the most recent search (S=5), and searches for them in **rname**.

*Example 3:*

```
S=7      <5>   Find (trip or tdbS) AND TExt=5.rname
```

finds all records containing either 'trip' or 'tdbs' *and* those **Text** field occurrences of **rname** terms from the records found by search result number five.



## Find – External Transaction Sets

### Description

This type of indirect search order applies the terms listed in an external file as search terms (linked by **OR**) to the database, and can be used in the same way as any other textual condition in a search order.

Single quotes enclosing the file name signify that only exact matches of that file's terms should be located.

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 path and file path name should be included. This is ordinary text searching, and the usual rules and defaults apply.

No **Define MAP** order is required in order to use external transaction sets. All the information required for the **Find** order is contained within it. The sole requirement for defining a transaction set is a filename.

### Syntax

```
Find [{field|fieldtype|view}=]FILE(file)
```

where *field*, *fieldtype* and *view* are the names of fields, field types and views to be searched, and *file* is the name of the file that contains the list of search terms.

### Examples

To work the examples below, create a text file, fill it with this information, entering one search term per line:

```
cup
bottle
dish
pot
kettle
cauldron
```

and save it as 'utensils.dat'. Its contents will be applied as search terms to the demonstration database **Alice**.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <25>  Find FILE(utensils.dat)
```

TRIP finds all of the records which contain any of the search terms stored in 'utensils.dat', constructing a search order of the form:

```
Find cup OR bottle OR dish OR pot OR kettle OR cauldron
```

*Example 2:*

```
S=3      <1>  Find TExt=FILE(utensils.dat) AND  
          chaptnr > 9
```

looks in all **TExt** fields of **Alice** for records containing any of the terms in 'utensils.dat', where the value for the **chaptnr** field is greater than nine.

*Example 3:*

```
S=4      <10> Find cook OR kitchen  
  
S=5      <2>  Find S=0 AND TExt=FILE(utensils.dat)
```

finds all of the records in the most recent search result (S=4) that contain any of the terms in 'utensils.dat' in any **TExt** field.

*Example 4:*

Create a text file containing 'white king' and 'white knight', and save it as 'person.dat'.

```
S=6      <55> Find PHrase=FILE('person.dat')
```

This order will then use exact matching to locate all exact occurrences of 'white king' or 'white knight' in any **PHrase** field.

## Find ABout

### Description

Whereas Boolean searching focuses on matching a request with a set of provably valid responses (using basic set theory), best match searching focuses on matching a request with a set of responses that reflect the underlying purpose or intent of the request, rather than any provably correct “answer”; from this set of possible matches, the algorithm then provides a weighting, or judgment, as to how relevant each matched document is when compared to the information need expressed in the query.

See the separate document `TRIP_White_paper_non_Boolean_search.pdf` for an in-depth description of non-Boolean searching in TRIP.

Before the `About()` function can be used, the databases to be searched must be enabled for non-Boolean search via the TRIPmanager administration program.

*Note:*

*For information on configuring the `About()` function, see the CCL command **Find ABout**.*

### Syntax

```
Find ABOut (term1 term2 term3 ...)  
Find ABOut (T=n [TO m])  
Find ABOut (R=n [TO m])  
Find ABOut (S=n [TO m])
```

### Examples

*Example 1:*

```
S=1      <10000>    BAsE tipster_small  
S=2      <27> Find about (peanut butter)  
S=3      <407> Find about (s=2)
```

## Find Class

### Description

This function finds records that have been assigned category tags, and can be combined within any Boolean expression as normal.

The search function accepts a string as argument, and this string is interpreted as the name of one or more categories within the current scheme (note that each open database may be assigned a different scheme and that this indirect mapping is performed for each open database separately). Once the names are matched, any category tags (record IDs) found are then located within the open database.

See the separate document [TRIP\\_White\\_paper\\_Classification.pdf](#) and the TRIPmanager help file for an in-depth description of document classification in TRIP.

Before the Class() function can be used, a Classification schema must be created and the databases to be searched must be enabled for category search. Both of these activities are done using the TRIPmanager administration program.

### Syntax

```
Find Class (name)
```

where *name* is a category name in a classification scheme setup via TRIPmanager.

### Examples

*Example1:*

```
S=1      <276> BAsE Ctest
S=2      <10> Find class(financial) and australia
S=3      <147> Find class(economic) and class(financial)
```

## Find SCoPe

*Note:*

*The “Find SCoPe” modifier is related to the “UPDate SCoPe” modifier and neither are connected with, or influenced by, the modifier “Define SCoPe”.*

### Description

This function is used to invoke previously saved, predefined search sets and to use them to search across large database clusters of mostly static data: For example, a cluster of databases containing mostly historical data the recent databases being the ones having data that changes more frequently.

In such a large database cluster, it may be useful to have several TRIP procedures – e.g. for different areas of interest – that are used to create pre-made search sets saved in a special SIF file. This file can then be used by TRIP only for searching, in order to simplify, standardize and speed up such searches.

### Syntax

```
Find SCoPe(procname1) AND (otherCriteria)
```

where *procname1* is the name of the pre-made search set and *otherCriteria* are any other search criteria needed for this particular search.

### Examples

*Example 1:*

```
S=1 base BCL=db01,db02,db03, ... db17,db18,db19,db20,db21
```

```
S=2 Find SCoPe(proc2) and (the other search criteria)
```

this search uses the pre-made search sets saved in the special SIF file and other criteria to perform a search on the open cluster.

*Note:*

*For more information, see **Appendix B - Scope Search Facility of the TRIPmanager Administration guide.***

## FRequency

### Description

This command gives information on the distribution of values within a given field in the records found by a particular search result.

### Syntax

```
FRequency [S=n] field {[r] TO r[,step]|step}}
```

or

```
FRequency [S=n] field[=mode[,step]]
```

where *n* is the number of any valid search, *field* is the name of the field whose contents are to be displayed by frequency, *r* is a field value that limits the range for which frequency is to be calculated, and *step* is the size of the interval in which the field values are to be grouped.

Range and *step* values may be defined for **NUmber** and **INteger** fields only. If the *step* value is omitted, TRIP will choose a suitable interval.

### Examples

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <8>   Find rname=dr
FRequency rname
```

opens demonstration database **Corr**, finds all records whose **rname** fields contain the word fragment 'dr', and presents a table of the hit terms and their frequencies for **rname**.

#### Example 2:

```
S=1      <99>  BAsE corr
S=2      <9>   Find PHrase=dr
FRequency rname
```

opens demonstration database **Corr**, finds all records whose **PHrase** fields contain the word fragment 'dr', and presents a table of the frequencies for hit terms in the **rname** field only.

#### Example 3:

```
S=1      <475> BAsE Alice
S=2      <100> Find R=TO 100
FRequency S=2 person
```

opens demonstration database **Alice**, finds the first one hundred records, and displays a frequency table listing the values contained in the **person** field and their frequencies.

#### Example 4:

```
S=1      <475> BAsE Alice
FRequency chaptnr 1 TO 8
```

makes a frequency analysis for chapter numbers between one and eight inclusive for database **Alice**.

*Example 5:*

To have the results of the previous search summarized into three bands or subgroups of two chapters each, add the **STEP** modifier to the command:

```
S=1      <475> BAsE Alice
S=2      <21>  Find mouse
FRequency S=2 chaptnr 1 TO 8, 2
```



## Frequency - Date/Time

### Description

The output from a **F**requency command can also be grouped for fields of type **DATE** or **TIME**.

### Syntax

```
FFrequency fld=level, interval
```

where

*level* = 1 or 2 or 3, indicating: 1=Years/Hours, 2=Months/Minutes, 3=Days/Seconds

*interval* = the interval within the level specified

The second argument, specifying the size of the interval will be rounded up to the nearest value if not dividable properly (specifying a minute interval to 17 will result in an interval of 20 which matches 60).

### Examples

#### Example 1:

To group the **F**requency command output of the time field **TIM** into intervals of 30 minutes, the command should be entered like this:

```
FFrequency TIM 2,30
```

Here the number 2 indicates minutes and 30 the interval.

This would generate output like this:

Value	Size	Per cent
00:00..29	1300	1.6
00:30..59	990	0.8
01:00..29	200	0.2

#### Example 2:

To group the **F**requency command output of the time field **TIM** into intervals of 2 hours, the command should be entered like this:

```
FReq TIM 1,2
```

Here the number 1 indicates hours and 2 the interval.

This would generate output like this:

Value	Size	Per cent
00..01	1300	1.6
02..03	990	0.8
04..05	200	0.2

**DATE** fields are handled in a similar way.



## FUZZ

*Note:*

*The CCL **FUZZ** order is not influenced by **DEfine FUZZ** and is not to be confused with the **FUZZ()** CCL modifier.*

### Description

**FUZZ** is similar to the Find command, except that it works in a manner similar to many Internet search engines and can thus be used instead of the **Find** command.

### Syntax

```
FUZZ term [term[...]]
```

where *term* is a search term and multiple terms are each separated by a single space.

By default, the use of multiple terms assumes a Boolean “AND” exists between each term; hence the default **FUZZ** command only returns records that include all given terms.

If it is required to locate a particular phrase, this can be achieved using double quotation marks around the phrase.

The default search behaviour can be overridden by adding Boolean operators, such as “AND”, “NOT”, “OR” and “XOR” between the desired search terms. Furthermore, the short forms for “AND”, “NOT” and “OR” (respectively, “+”, “-” and “|”) will function correctly; their actual function being dependant on how they are placed (See table 18).

*Note:*

*In the following table, special attention should be given to the placing of the separating spaces for each entry in the example column.*

**Table 18 - Use of Short form operators with the Fuzz command**

Short form	Operation	Example	Returns records containing:
+	AND	FUZZ AAA + BBB	AAA and BBB
+	AND	FUZZ AAA +BBB	AAA and BBB, where BBB can be a stop word (See below)
	OR	FUZZ AAA   BBB	AAA or BBB
-	NOT	FUZZ AAA - BBB	AAA not BBB
-	NOT	FUZZ AAA -BBB	AAA not BBB
-	Unary NOT	FUZZ -AAA	everything <i>excluding</i> AAA
-	Unary NOT	FUZZ -AAA +BBB	everything <i>excluding</i> AAA but <i>including</i> BBB, where BBB can be a stop word (See below)

*Note:*

*See also, **DEfine WEIght** for how to adjust field weighting.*

### Stop-words

FUZZ commands can be set up to ignore common words and characters as they tend to slow down your search without improving the results (See Define Stop Word). TRIP will indicate if a common word has been excluded by displaying the omitted terms in a warning message.

If common words are stop words, but are essential to obtaining the required results, they can be included by either enclosing them in double quotation marks, or by putting a "+" sign immediately in front of each word required. (Be sure to include a space before the "+" sign.)

## Examples

### Example 1:

```
S=1      <475> BAsE Alice
S=2      <1>   FUZz mome rath
```

returns only those records containing the search terms "mome" AND "rath"

### Example 2:

```
S=1      <475> BAsE Alice
S=2      <15>  FUZz "down the rabbit hole"
```

returns only records containing the exact phrase "down the rabbit hole".

### Example 3:

```
S=1      <475> BAsE Alice
S=2      <14>  FUZz "down the rabbit hole" + alice
```

returns only records containing the term "alice" AND the exact phrase "down the rabbit hole", even if some of the words enclosed by quotation marks are stop words.

### Example 4:

```
S=1      <475> BAsE Alice
S=2      <3>   FUZz mome +the raths
```

returns records containing the terms "mome" AND "rath", as well as the stop word, "the".

### Example 5:

```
S=1      <475> BAsE Alice
S=2      <3>   FUZz mome | rath
```

**FUZz** finds all records that contain either the term "mome" OR the term "rath".

### Example 6:

```
S=1      <475> BAsE Alice
S=2      <2>   FUZz mome - rath
```

**FUZz** returns only records containing the term "mome" but NOT the term "rath".

### Example 7:

```
S=1      <475> BAsE Alice
S=2      <2>   FUZz mome -rath
```

Without the space separating '-' and 'rath', gives the same result as the example above.

### Example 8:

```
S=1      <475> BAsE Alice
S=2      <472> FUZz -mome
```

**FUZz** returns all records, *except* those containing the search term mome.

**Example 9:**

```
S=1      <475> BAsE Alice
```

```
S=2      <424> FUZz -mome +alice
```

**FUZZ** returns all records containing the stop word “alice”, *except* those containing the search term mome.



## Help

### Description

**Help** provides assistance on commands, symbols, modifiers and qualifiers used as modifiers in search orders, and database names and fields for the TRIP demonstration databases.

Below is a list of items found in TRIP's help texts, classified by function and then alphabetized. Although the shortest form of many TRIP commands is a single character, **Help** in most cases requires you to type at least the first three characters of the term for which you need assistance. The upper-case characters represent the minimum number of letters TRIP will accept for each item.

### Commands

Table 19 – Commands for which online help is available

BACK	EXPORT	INSERT	SELECT
BASE	FIND	MEASURE	SFORM
CALI	Find?	MORE	SHOW
CONTINUE	FREQUENCY	NEXT	STATUS
DEFINE	FUZZ	PREVIOUS	STOP
DELETE	HELP	PRINT	TOP
DISPLAY	HIDE	REVEAL	TRACE
EDIT	IMPORT	RUN	UPDATE
EXPAND	INDEX	SAVE	

### Operators

Table 20 – Operators for which online help is available

AND	AND.T	NOT.E
AND.C	AND.W	NOT.F
AND.E	OR	NOT.P
AND.F	XOR	NOT.S
AND.P	NOT	NOT.T
AND.R	NOT.C	NOT.W
AND.S		

### Symbols

Table 21 – Symbols for which online help is available

\$	.	,
\$ \$ \$...\$	...	>
#	.....	<
:	...	
!	&	

The ellipsis [...] signifies 'an infinite number of', and is not to be included as part of the CCL **Help** order. The [...] symbol represents a single space.

## Modifiers/Qualifiers

Table 22 – Modifiers & qualifiers for which online help is available

ACCess	FORmat	NT	SORt
ALL	FRequency	NUMber	SPAcE
BASe	FRom	PAGe	SUBField
BT	FUZz	PARagraph	T
COMmand	GROup	PART	TEXT
CONtinue	HIGHlight	PCode	TForm
COPy	HOLD	PHRase	THESaurus
COST	INSert	PRInt	TIME
CT	KEY	PRINTER	TIMEForm
CUT	LOCal	PROcedure	TO
DATE	LPCode	R	TSTamp
DEScending	MAP	REVerse	UP
DISplay	MASK	RT	USEr
DOWN	MAXimum	S	VIEW
EForm	MERGE	SCOpe	WAIT
FIeld	MINimum	SDI	WEIght
FILE	NO	SENTence	WHEre
FINd	NOW	SForm	WORD
FOCus			

## Syntax

Help [{command|symbol|modifier|operator|qualifier}]

where *command*, *symbol*, *modifier*, *operator* and *qualifier* are any of the items presented in the preceding tables.

## Examples

*Example 1:*

Help

brings up instructions on using general help. These include lists of the commands and symbols available in the **Help** database, with a short description of each item.

*Example 2:*

Help Find

or just

H Fin

gives help on the **Find** command.

## HIDe (TRIPclassic only)

### Description

This command is used in procedures to prevent CCL commands from echoing to the screen during execution. It has no effect during manual CCL processing. The command **REVEal** restores CCL output to screen. Neither command accepts arguments.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

HIDe



## IMPORT

### Description

**EXPORT** is used to extract the contents of a 'Control' record and write it into a text file, and its companion command **IMPORT** creates or updates a 'Control' record using the contents of this file. This allows database, form and format designs to be copied from one 'Control' file to another. Only the owner/creator of the item to be **EXPORTed** (database design, form, format etc.) may **EXPORT** it.

If any of the records to be copied require editing, do this within TRIP rather than within the text file.

If the record specified in an **IMPORT** order already exists in the destination 'Control' file, add the **UPDATE** modifier to the **IMPORT** order. The imported record then replaces the original one.

Format and entry form names consist of the name of their parent database, a full stop or period and the name of the format itself, for example, **Corr.corr\_full**, or **Alice.1**.

### Syntax

```
IMPORT [UPDATE] {BASE|Format|EForm|SForm|THESaurus  
|PROCEDURE}=item PATH=filePath FILE=file
```

where *item* is the name of the host or destination **BASE**, **Format**, **EForm**, **SForm**, **THESaurus** or **PROCEDURE** to be imported, *filePath* is the path to which the BAF, BIF and VIF files will be saved and *file* is the holding file from which this information will be copied.

*Item* can also be *base.\**, which specifies that all database elements (database design, output formats and data entry forms) be imported.

### Examples

#### Example 1:

```
EXPORT BASE=Alice FILE=alice.def  
IMPORT BASE=wonder FILE=alice.def
```

exports the database design for the demonstration database **Alice** into a file called 'Alice.def', then imports or copies it to a file named 'Wonder'.

#### Example 2:

```
EXPORT BASE=Alice FILE=alice.def  
IMPORT UPDATE BASE=mirror FILE=alice.def
```

exports the database design for the demonstration database **Alice** into a file called 'Alice.def', then updates or overwrites the database design named **Mirror**.

#### Example 3:

```
EXPORT Format=alice.1 FILE=alice.fmt  
IMPORT Format=wonder.1 FILE=alice.fmt
```

exports an output format called '1' for database **Alice** into a file called 'Alice.fmt', then imports the file into the format called '1' for the database **Wonder**.

*Example 4:*

```
EXPort PRocedure=proc FILE=transfer.pro  
IMPort PRocedure=public.proc FILE=transfer.pro
```

exports a procedure called 'Proc' into a file called 'transfer.pro', then imports it to a procedure named 'Public.proc', to which all members of the group 'Public' have access. Only the procedure's owner may export that procedure.

*Example 5:*

```
EXPort EForm=alice.1 FILE=alice.ef1  
IMPort EForm=new_alice.1 FILE=alice.ef1
```

exports a data entry form called '1' for database **Alice** into a file called 'Alice.ef1', then imports it to a data entry form named '1' for database **New\_alice**.

*Example 6:*

```
EXPort BAsE=Alice.* FILE=alice.def  
IMPort BAsE=alice_copy.* FILE=alice.def
```

exports all elements of the database **Alice** (formats and entry forms included) and imports them to database **Alice\_copy**.

*Example 7:*

```
EXPort SForm=alice_demo FILE=alice_demo.sfo  
IMPort SForm=alice_search FILE=alice_demo.sfo
```

exports the search form called 'Alice\_demo' to a file named 'Alice\_demo.sfo', then imports the file to search form 'Alice\_search'.



## INDEX

### Description

This command submits an index job for the database specified. Write access to the specified database is necessary in order to index it.

### Syntax

```
INDEX database
```

where *database* is the name of the database to be indexed.



## INsert

### Description

**INsert** is a global update command used to insert new material into one or more selected records simultaneously, using a single order. The new data may take the form of words, sentences, paragraphs or subfields, which may be inserted into destination records using the search hit position or field name. As with every other command capable of altering data, **INsert** requires write access to the database in question. Double quotes surrounded strings to be inserted are strongly recommended.

When in doubt regarding the positions referenced in a search, first display the records using **Show**—the highlighted terms will occupy the positions in question.

The qualifier **Where** states which search results should be updated. When followed by **S=** and a search result number material will be added to the records of only that search; when used with **R=** and a record number or list or range of record numbers, only those records will be modified. Specify ranges of record numbers with **TO** and **FRom**. Ranges and values are separated by commas.

**INsert COPY** allows copies of selected records to be modified, then added to the current (or other) database.

Refer to the 'EDit' section of this manual for further information regarding **EDit INsert**, a method of adding new data using a data entry form.

### Syntax

Inserting using search results:

```
INsert [COPy[=database]] {SUBField|PARagraph
|SENTence|WORD|field[.{subfieldno
|paragraphno[.sentenceno]}}} =newdata WHere S=n
```

Inserting using record numbers:

```
INsert [COPy[=database]] field[.{subfieldno
|paragraphno[.sentenceno]}}] =newdata
WHere R=range
```

where *database* is the name of the database to receive new data, *field* the name of the field and *subfieldno*, *paragraphno* and *sentenceno* the numbers of the subfields, paragraphs or sentences to receive new data, *newdata* the new material to be inserted, *n* a record or search result number and *range* represents a record number, list of record numbers or range of record numbers.

The modifiers are restricted by field type, as shown below:

**Table 23 – INsert order modifier restrictions by field type**

Modifier	Applicable Field Type
SUBField	PHrase only
PARagraph	TExt only
SENTence	TExt only
WORD	TExt and PHrase

## Examples

Examples one through six are taken from demonstration database **Corr**, and are intended to be worked sequentially. Examples four through six operate on record numbers throughout the database.

### Example 1:

```
S=1      <99>  BAsE corr
S=2      <60>  Find mats
INSert WORD="John" WHere S=0
```

inserts the word 'John' in front of each word of every **TEx**t and **PH**rase field of every record located by the most recent search.

### Example 2:

```
S=3      <5>   Find mats and result#
INSert SENTence="I told you so!" WHere S=3
```

inserts 'I told you so!' before each sentence found by search result three.

### Example 3:

```
INSert SUBField="John Smith" WHere S=3
```

inserts the new subfield 'John Smith' before each subfield found by search number three.

### Example 4:

```
INSert rcomp="TSI" WHere R=FRom 75
```

appends a new subfield containing 'TSI' at the end of the field **rcomp** in all records numbered seventy-five or higher.

### Example 5:

```
INSert rcomp.1="TSI" WHere R=70 TO 74
```

inserts 'TSI' as a new *first* subfield into the field named **rcomp** in records seventy through seventy-four inclusive. Existing data will be moved forward by one subfield.

### Example 6:

```
INSert content.3.1="I told you so!" WHere R=FRom 95
```

inserts 'I told you so!' as a new *first* sentence to the third paragraph of the **TEx**t field **content** in all records numbered ninety-five and above. Existing sentences are moved one step forward.

### Example 7:

```
INSert COpy=corr rname="Director of Marketing" WHere S=3
```

defines **Corr** as the copy destination database, then appends 'Director of Marketing' to the end of each **rname** field in search result three as a new subfield. It then copies the modified records back to **Corr**, leaving the original records unchanged.

**Example 8:**

```
DEfine COPy=corr  
S=4      <1>   f scountry=irak  
INSert COPy rname="Marketing Manager" WHere S=0
```

designates the copy destination database as **Corr** using **DEfine COPy**. It then appends the new subfield 'Marketing Manager' to the end of the **rname** field of the record containing 'Irak' in the field **scountry** in the latest search result (S=4), and appends the modified record to **Corr**. *Only the copied record is modified*—the original record is unchanged.



## LEAve (TRIPclassic only)

### Description

A **LEAve** order takes you from the CCL window to the menu of TRIPclassic.

*Note:*

*This command is valid in TRIPclassic only and has the same effect as pressing the key PF3 on the keypad.*

### Syntax

To leave the CCL window.

LEAve



## List

### Description

**List** is equivalent to **Find?**, in that it displays a complete list of all search results compiled during the current search session. This command displays the search history from the beginning, whereas summary lines for the last five search results are always given in the CCL history window while searching is in progress.

Use the <↑ **Arrow**> and <↓ **Arrow**> keys or the **More** and **Back** commands to scroll the list.

### Syntax

List



## LOAd

### Description

A **LOAd** order can be used to batch load TFORM data into a Trip database.

### Syntax

To load TFORM data.

```
LOAd [INDeX] dbname TForm=filename [[NO] SENTence]  
[TStamp=YYYYMMDD [HH:MM:SS]]
```

*INDeX* means that indexing will take place after the loading.

*SENTence* means that sentence and paragraph scanning will take place - this is the default.

*TStamp* means that if the records in the *TFORM* file have time stamps then only records whose time stamp is later than the one given in the command will be loaded into the *BAF* file; the date part of the time stamp may be given either in the generally accepted format (as above) or in the format defined for the session when the command is given. This modifier is unlikely to be used for other *TFORM* files than those created automatically by TRIP as log files for the database (if the creator has required such a file in the database design) - and then only if the database manager wishes to load only some of the records from the file.

### Examples

*Example 1:*

```
LOAd corr TForm=c1.tfo
```

will take the file *c1.tfo* and try to load its contents into the database *corr* - it will, in fact, have the same effect as using the *Load* menu option from inside TRIPclassic.

*Example 2:*

```
LOAd INDeX corr TForm=c1.tfo
```

will also index the database after the loading of the *TFORM* file. It will, in fact, have the same effect as using the *Load and Index* menu option from inside TRIPclassic.

## MEasure

### Description

**MEasure** may be applied to both **NUmber** and **INteger** field types in the records of a search result. It generates common statistical measures such as average and standard deviation for a single **NUmber** or **INteger** field, and presents them in a table in the **Show** window.

If a value contains too many digits, it will be replaced by the overflow symbol [#].

### Syntax

```
MEasure [S=n] field
```

where *n* is the number of any valid search, and *field* is the name of the field whose contents are to be statistically measured.

### Examples

#### Example 1:

```
S=1    <24>  BAsE carroll
S=2    <12>  Find wood
S=3    <6>   Find S=2 AND tree
MEasure chaptnr
```

This series opens the **Carroll** database (S=1), finds all of the records which contain the term 'wood' (S=2) as well as those containing both 'wood' and 'tree' (S=3), and gives a statistical analysis for the **INteger** field **chaptnr** of the most recent search result (S=3).

#### Example 2:

```
S=1    <24>  BAsE carroll
S=2    <12>  Find wood
S=3    <6>   Find S=2 AND tree
MEasure S=2 chaptnr
```

This series opens the **Carroll** database (S=1), finds all of the records which contain the term 'wood' (S=2) as well as those containing both 'wood' and 'tree' (S=3) and gives a statistical analysis for the **INteger** field **chaptnr** of the specified search result (S=2).



## More

### Description

This order is used similarly to the <↓ **Arrow**> key on the keyboard to advance either one window or a specified number of lines in the displays provided by commands such as **Show**, **Display**, **Find?**, **Help**, **List**, and **Status**. If **More** is used after a **Show FOCUS** order, it focuses on the hit following the one currently shown.

If **Show** is used with a page-oriented output format (that is, one containing form feeds, or header or trailer boxes), the number of lines argument *n* is not allowed.

TRIP provides a message if the number of lines specified exceeds the number of lines remaining in the output.

### Syntax

More [*n*]

where *n* is the number of lines to move forward.

## Next

### Description

This order moves forward a specified number of records to the top of the next record in a display. If it is used after a **Show Focus** order, it focuses on the next hit.

TRIP provides a message if the number of lines specified exceeds the number of lines remaining in the output.

### Syntax

```
Next [n]
```

where  $n$  is the number of records to move forward.



## PREVIOUS

### Description

This command is the opposite of **Next**, causing **Show** output to move backward a specified number of records, to the top of the previous record.

### Syntax

```
PREVIOUS [n]
```

where *n* is the number of records to move backward.



## Print

### Description

A **Print** order sends records to the printer queue, or, if the order contains a file specification, writes them to a text file.

There are several types of **Print** orders, including those that print the results of searches and others that print information about the system. CCL **Print** orders print to a file on the host or a printer connected to the host, with the exception of **Print Local**, which sends output to the local printing device.

**Print** and **Show** orders have many modifiers and qualifiers in common. In most cases, **Print** and **Show** orders can be used identically to specify search results, formats, sorting, record numbers and so forth. They also share the same format and search result defaults.

### Print – Section Index

Print orders are covered under the following headings:

Table 24 – List of Print orders

Command	Subject
Print	A Simple Order
Print	ACcess
Print	BASe
Print	Display
Print	EForm
Print	FILE
Print	FOcus
Print	Format
Print	FRequency
Print	GRoup
Print	Hlghlight
Print	Llst
Print	Local
Print	MEasure
Print	PART SORT
Print	PRINTER
Print	PRocedure
Print	Record
Print	REVerse
Print	Search Result
Print	SForm
Print	SORT
Print	TForm
Print	Timing
Print	TRace
Print	USer

## Syntax

To print the contents of a search result:

```
Print {Local|[{NOW|WAIT|[NO]HOLD}]}
      [FILE=file][PRINTER=printcontrolfile]
      [{S=n|BASE=database}][R=range]
      [Format={format|{field|fieldtype|view}
      [, {field|fieldtype|view}[,...]]}]
      [{[NO]REVERSE|[PART]SORT={field[DESCending]
      [,field[DESCending][,...]]|FREQUENCY}[[NO]MERGE}]]
      [FOCUS][HIGHLIGHT]
```

To print all or part of the search history:

```
Print [{NOW|WAIT|[NO]HOLD}]
      [FILE=file][PRINTER=printcontrolfile]
      {LIST|TRACE[S=range]}
```

To print a list of terms:

```
Print [{NOW|WAIT|[NO]HOLD}]
      [FILE=file][PRINTER=printcontrolfile]
      Display[{field|fieldtype|view}=]term
      [S=n][SORT=FREQUENCY]
```

To print fuzzy displays:

```
Print [{NOW|WAIT|[NO]HOLD}]
      [FILE=file][PRINTER=printcontrolfile]
      Display [{field|fieldtype|view}=]FUZZ
      (term[,percentage][,depreciated]
      [,find-terms][,algorithm])
      [S=n][SORT=FREQUENCY]
```

To print terms from a thesaurus:

```
Print [{NOW|WAIT|[NO]HOLD}]
      [FILE=file][PRINTER=printcontrolfile]
      Display {CT|BT|NT|RT|UP|DOWN} (term)
```

To print information regarding databases, forms, etc.:

```
Print {Local|[{NOW|WAIT|[NO]HOLD}]}
      [FILE=file][PRINTER=printcontrolfile]
      [{BASE[LIST]|[BASE]ACCESS|PROCEDURE|
      GROUP|USER}[R={item|ALL}]|SFORM [R=item]|
      {EFORM|FORMAT}[{BASE=database|R=ALL}]]
```

To print statistics:

```
Print [{NOW|WAIT|[NO]HOLD}]
      [FILE=file][PRINTER=printcontrolfile]
      {FREQUENCY[S=n]field[{[r] TO r[,step]|step]}|
      MEASURE[S=n]field}
```

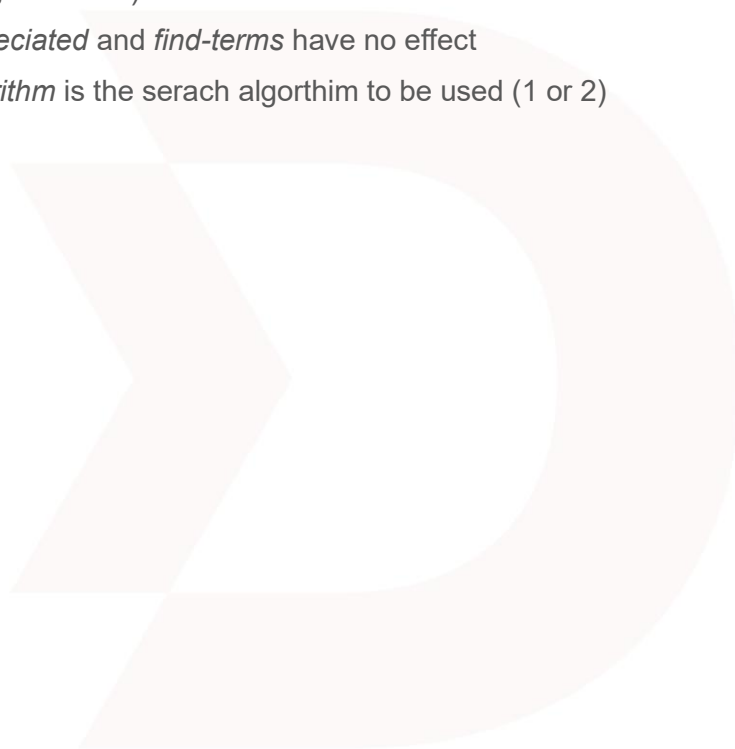
To print to a TForm file:

```
Print [{NOW|WAIT|[NO]HOLD}] [R] TFORM=file
```

where:

1. *file* is the name of a file to which to print
2. *printcontrolfile* is a printer control file

3. *n* is a search result number
4. *range* is a single record or search result number, list of record or search result numbers or range of record or search result numbers
5. *format* is the name of an output format to be used
6. *field*, *fieldtype* and *view* are fields, field types and views whose contents are to be printed
7. *item* is the name of a database, search form, procedure, user or group
8. *R* is a field value that limits the range for which frequency is to be calculated
9. *step* is the size of the interval in which the field values are to be grouped
10. *term* is a search term
11. *percentage* is the desired percentage of word matching (range: 1...100)
12. *depreciated* and *find-terms* have no effect
13. *algorithm* is the search algorithm to be used (1 or 2)



## Print – A Simple Order

### Description

If a **Print** command is given alone, the most recent search result (**S=0**) is output in whatever default output format has been defined for the database, without sorting or focus, and only after the TRIP session has ended.

### Syntax

```
Print[?]
```

### Examples

*Example 1:*

```
Print
```

stores a **Print** request for the latest search result and sends it to the printer queue. It will be printed when the requester exits TRIP.

*Example: 2*

```
Print?
```

lists any **Print HOLD** requests made during the current session.

## Print ACcess

### Description

This order enables database administrators and the user SYSTEM to print access information for users of their databases. Databases are presented in alphabetical order, while users having access are listed chronologically by access date.

The addition of the *database* construct limits the screen output to the specified database only. **Print ACcess** is equivalent to **Print BAsE ACcess**.

### Syntax

```
Print [BAsE] ACcess [R={database|ALL}]
```

where *database* is the database whose status is requested. **Print ACcess** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

#### Example 1:

```
Print ACcess
Print BAsE ACcess
```

Both of these orders output the read and write privileges of each user and user group, for all the databases that the database administrator has created. The databases are arranged alphabetically, and the user and user groups listed in chronological order by date of creation.

#### Example 2:

```
Print ACcess R=Alice FILE=alice_acc.dat
Print BAsE ACcess R=Alice FILE=alice_acc.dat
```

Both orders print the read and write access rights for database **Alice** to the file called 'alice\_acc.dat', but only if the database in question has been created by the manager making the request. SYSTEM may output access information for any database.

#### Example 3:

```
Print ACcess R=ALL NO HOLD
Print BAsE ACcess R=ALL NO HOLD
```

Both orders print the access rights for all users and user groups of all databases on the TRIP system immediately. The **ALL** modifier is available only to user SYSTEM.



## Print BAsE

### Description

This order prints information regarding databases.

**Print BAsE** without modifiers prints complete status listings for all databases that the user has created. **Print BAsE LIst** prints a less-detailed list of databases to which the user has read-access. **Print BAsE R=database** prints a status listing only for the specified database. **Print BAsE=database** prints all of the records of the specified database.

### Syntax

To print status screens for or information about databases:

```
Print BAsE [LIst] [R={database|ALL}]
```

where *database* is the name of the database whose status is to be printed. **Print BAsE** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

To print all of the records of a database:

```
Print BAsE=database
```

where *database* is the name of the database from which records are to be printed. **Print BAsE=** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=**, **TForm=**, **S=**, **R=**, **Format=**, **REVerse**, **SORt=**, **FOcus** and **HLighT** modifiers.

### Examples

*Example 1:*

```
Print BAsE
```

prints the **STatus** lists for all databases which the database administrator has created, in alphabetical order by database.

*Example 2:*

```
Print BAsE R=corr
```

prints the status screens for the demonstration database **Corr**.

*Example 3:*

```
Print BAsE R=ALL
```

prints the status screens for all databases on the TRIP system, and is available only to the user called SYSTEM.

*Example 4:*

```
Print BAsE LIst
```

prints a list of all databases to which the user has read-access, with their owners and descriptions.

*Example 5:*

```
Print BAsE LIst R=corr
```

prints the name, owner and description of the demonstration database **Corr**.

*Example 6:*

```
Print BAsE LIst R=ALL
```

prints the name, owner and description of every database on the TRIP system, and is available only to the user called SYSTEM.

*Example 7:*

```
Print BAsE=corr
```

prints all of the records of demonstration database **Corr**, in the order in which they were added to the database.

*Example 8:*

```
Print BAsE=corr R=1 TO 6 NO HOLD
```

prints the first six records of demonstration database **Corr** immediately, in the order in which they were added to the database.



## Print Display

### Description

This order prints the result of a **Display** list.

Refer to the sections on 'Display' in this manual for a complete list of **Display** options.

### Syntax

```
Print Display [{field|fieldtype|view}=]term
```

where *field*, *fieldtype* and *view* are the names of a field, field type or view to be print-displayed and *term* is the desired search object. **Print Display** can also be used with the **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=** and **S=** modifiers, as well as **SORT=** **FRequency**, **FUZZ** and the thesaurus modifiers.

### Examples

All of these examples use the **Text** and **Phrase** search field type default for demonstration database **Alice**.

#### Example 1:

```
Print Display mou#
```

prints the display list for **Text** and **Phrase** field terms beginning with 'mou' when the user exits TRIP.

#### Example 2:

```
Print Display speaker=m# NO HOLD
```

prints the display list for all terms in the **speaker** field beginning with 'm' immediately.

#### Example 3:

```
Print Display S=2 al# FILE=al_terms.dis
```

prints the display list of all **Text** and **Phrase** field terms beginning with 'al' in search result number two to a file called 'al\_terms.dis' when the user exits TRIP.

#### Example 4:

```
Print Display S=3 mou# SORT=FRequency NO HOLD
```

prints a display list of all **Text** and **Phrase** field terms beginning with 'mou' in search result number three, sorted by number of occurrences, immediately.

#### Example 5:

```
Print Display FUZZ(mouse) FILE=mouse.dis NOW
```

prints a fuzzy display list of all **Text** and **Phrase** field terms resembling mouse to a file called 'mouse.dis' immediately.

## Print EForm

### Description

This order prints a list of all the TRIPclassic entry forms accessible to the user. The addition of the *database* construct limits the output to forms of the specified database only.

### Syntax

```
Print EForm [{BAsE=database|R=ALL}]
```

where *database* is the name of the database for which data entry form information is to be printed. **Print EForm** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
Print EForm
```

prints a list of the entry forms belonging to all databases owned by the user, in alphabetical order by database name.

*Example 2:*

```
Print EForm BAsE=Alice
```

prints only the information for all entry forms created for use with demonstration database **Alice**.

*Example 3:*

```
Print EForm HOLD BAsE=corr FILE=eformfile.dat
```

prints a list of data entry forms available for the demonstration database **Corr** to a file called 'eformfile.dat' after the TRIP session has ended.

*Example 4:*

```
Print EForm R=ALL
```

prints information for all entry forms, and may only be used by the user called SYSTEM.

## Print FILE

### Description

Where output is to be imported by another software package such as a word processor or spreadsheet, it is often useful to print the output from TRIP to a file rather than a physical device.

The structure of the complete file specification is host-dependent, and may include an optional full logical or physical pathname.

### Syntax

```
Print [{NOW|WAIT|[NO]HOLD}] FILE=[{'|"}][path]filename
[{'|'"}]
```

where *path* is the (optional) path to *filename* and *filename* is the name of the file where the information will be stored.

*Note:*

*File names (with or without path strings included) that contain spaces must be enclosed in single (') or double (") quotes.*

**Print FILE** can be used with any other **Print** modifier except **TForm=** and **Local**.

*Notes:*

- *All PRINT output will be processed by the TRIP server machine and not by the client machine*
- *If printing from within TRIPmanager on the Windows server where TRIP is installed, you MUST use the NOW modifier or nothing will be printed*
- *PRINT FILE overwrites existing files of the same name with NO WARNINGS.*
- *If the file path is incorrect, or if access rights to the location for the file to be printed are incorrect, a 'file could not be opened' error is appended to the print instruction in the PRxxxxx.log file.*
- *If no path is specified, the file will be printed to the current process' directory. For example, on Windows 7:*
- *for the 'TRIP classic' start menu shortcut, this will be the <TRIPinstallation>\bin directory*
- *for a TRIPclassic session started in a 'cmd' window, this will be the directory the session was started in*
- *for TRIPmanager this will be the \windows\system32 directory*

### Examples

These are taken from the demonstration database **Alice**.

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <4>   Find R=1 TO 4
Print FILE=alicetext
```

writes the records located by the most recent search result to a file called 'alicetext'. The file extension is optional.

*Example 2:*

```
S=3      <426> Find Alice
S=4      <29>  Find AND cat OR watch
Print S=3 FILE=mysearch.dat
```

writes the records found by search number three to the file 'mysearch.dat'.

**For a Windows host:**

*Example 3:*

```
Print FILE= C:\Users\Fred\Documents\TRIPdata\myfile.dat
```

prints the results of search number four to a file called 'marc.wiz' using the path name C:\Users\Fred\Documents\TRIPdata\. This print request will be executed when the TRIP session is ended.

**For a UNIX host:**

*Example 4:*

```
Print S=4 FILE=/usr/users/dompio/marc.wiz
```

prints the results of search number four to a file called 'marc.wiz' using the path name /usr/users/dompio. This print request will be executed when the TRIP session is ended.

*Example 3:*

```
Print S=3 NOW FILE="C:\Users\Fred\My Documents\TRIPdata\My
New File.txt"
```

prints the results of search number three to a file called 'My New File.txt' using the path name C:\Users\Fred\My Documents\TRIPdata\. This print request will be executed immediately.

## Print FOCUS

### Description

This command outputs only the paragraph of a **Text** field or subfield of a **Phrase** field which contains the desired hit.

The name of the field containing the hit term and the record number within the database in which it is found are shown in the output.

### Syntax

```
Print FOCUS
```

Print FOCUS can also be used with the Local, NOW, WAIT, [NO] HOLD, FILE=, PRINTER=, S=, R=, Format=, REVERSE, SORT= and Highlight modifiers.

### Examples

These examples use demonstration database **Alice**.

#### Example 1:

```
S=1      <475> BASE Alice
S=2      <10>  Find watch
Print FOCUS
```

prints only the hit paragraph or subfield from the results of the most recent search, in the default format.

#### Example 2:

```
S=3      <34>  Find white rabbit
S=4      <1>   Find and waistcoat
Print FOCUS Format=short S=3
```

prints only the hit paragraph or subfield from search result number three in the format called 'short'.

#### Example 3:

```
S=5      <1>   Find new zealand
Print FOCUS NO HOLD PRINTER=lpt1
```

prints only the hit paragraph or subfield from the most recent search result (S=5) immediately using the printer control file called 'LPT1.PRN'.

## Print Format

### Description

Selected information can be printed using pre-defined or run-time output formats.

Pre-defined 'named' formats can be created by anyone with read access to the desired database, and those available for any database can be seen in the **Status** list. Usually one of these formats is set as the default; this can be changed using the **Define Format** order. For more information, refer to the previous section covering the **Define** order in this guide.

Run-time formats can be defined directly in the **Print** order, and are essentially a list of the field names, field types and/or views (separated by commas) to be output. The available field names and types for any database can be seen in the **Status** list. If a particular combination of field names and/or types is used frequently, it can be defined as the session default using a **Define** order.

Although the three types of field qualifier, field name, field type and view can be mixed in the same **Print** order, the two types of format, named and run-time, cannot.

This order can also be used to print a list of all the output formats for databases owned by the user. The addition of the *database* construct limits the output to formats of the specified database only.

### Syntax

For named formats:

```
Print Format=format
```

For run-time formats:

```
Print Format={field|fieldtype|view}
[, {field|fieldtype|view}[,...]]
```

**Print Format=** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=**, **S=**, **R=**, **REVERSE**, **SORT=**, **FOCUS** and **HIGHLIGHT** modifiers.

To print available output formats:

```
Print Format [{BASE=database|R=ALL}]
```

where *format* is the name of a pre-defined format to be used, *field* *fieldtype* and *view* are the names of the fields, field types or views to be output and *database* is the name of the database for which usable output formats will be displayed. **Print Format** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
S=1      <475> BASE Alice
S=2      <11> Find R=5 TO 15
Print Format=chaptnr,person
```

This run-time format prints the contents of the fields **chaptnr** and **person** of the most recent search result (S=2), where these are not empty. The content is preceded by the field name, and output order of the fields is the same as in the **Print** order; i.e. in this case **chaptnr** appears before **person**.



**Example 2:**

```
S=3      <20> Find dormouse
S=4      <3> Find and white rabbit
Print HOLD Format=chaptnr,PHrase S=3
```

This run-time format prints the results of search result number three, printing the contents of the fields **chaptnr** and all **PHrase** fields where these are not empty. The output order of the fields is the same as in the **Print** order, with the output order of the **PHrase** fields as they are on the database, and the print job will execute after the current TRIP session has ended.

**Example 3:**

```
S=5      <99> BAsE corr
S=6      <2> Find prospects
Print Format=DAte,TEText SORT=day DESCending
```

prints the contents of all non-empty **TEText** and **DAte** fields in the latest search result, sorted in descending order by the contents of the **day** field and headed by their field names.

**Example 4:**

```
S=7      <1> Find and rcountry=england
Print Format=2
```

prints the results of the most recent search (S=7) in the predefined format named '2'.

**Example 5:**

```
S=8      <475> BAsE Alice
S=9      <1> Find australia
Print Format=dump
```

prints the contents of *all* of the fields of the database for the most recent search result (S=9), where these are not empty. The content is preceded by the field name, and the field output order is according to type: **PHrase**, **Number**, **Integer**, **DAte**, **Time** and **TEText**.

'Dump' is a pre-defined format supplied with TRIP.

**Example 6:**

```
Print Format
```

prints a list of all of the formats for databases that the user owns.

**Example 7:**

```
Print Format BAsE=Alice
```

gives a list of all of the predefined formats for the demonstration database **Alice**. This order may be given by any user with read access to the database in question.

**Example 8:**

```
Print Format R=ALL
```

prints a list of all of the named output formats on the TRIP system, and is available only to the user called **SYSTEM**.

## Print FRequency

### Description

This order outputs the result of a **FRequency** command.

### Syntax

```
Print FRequency [S=n] field [{[r] TO r[,step]|step}]
```

where *n* is the number of any valid search, *field* is the name of the field whose contents are to be displayed by frequency, *r* is a field value that limits the range for which frequency is to be calculated, and *step* is the size of the interval in which the field values are to be grouped.

Range and *step* values may be defined for **NUMber** and **INteger** fields only. If the *step* value is omitted, TRIP will choose a suitable interval.

**Print FRequency** can also be used with the **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=** and **S=** modifiers.

### Examples

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <60>  Find mats
Print FRequency rname
```

opens the demonstration database **Corr**, locates all occurrences of the name 'mats' and prints the frequency table for the field **rname** for the most recent search result (S=2).

#### Example 2:

```
S=3      <12>  Find dr
Print FILE=doctors.fre NO HOLD FRequency rname
```

locates all occurrences of 'dr' and prints the frequency table for the field **sname** immediately to a file named 'doctors.fre'.

#### Example 3:

```
S=4      <475> BAsE Alice
S=5      <21>  Find mouse
Print FRequency chaptnr 1 to 8, 2
```

locates all **TEXT** and **PHrase** field occurrences of 'mouse' in database **Alice** and prints a frequency table summarizing the **chaptnr** field into three bands or subgroups of two chapters each.

## Print GRoup

### Description

The **GRoup** modifier requests information regarding TRIP user groups in the system, and is available only to user managers and the user named SYSTEM. A user manager can print only information regarding his or her groups, whereas SYSTEM can print data for all groups.

### Syntax

```
Print GRoup [R={group|ALL}]
```

where *group* is the name of the group for which information is being requested. **Print GRoup** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
Print GRoup
```

prints a list of group members and their access privileges for all groups created by the user manager giving the order.

*Example 2:*

```
Print GRoup R=sales_teams FILE=teams.fil
```

may be used only by the creator of the group called 'Sales\_teams' or user SYSTEM, and prints the members and member privileges of that group to a file called 'teams.fil' upon leaving TRIP.

## Print Highlight

### Description

Hit terms within printed output, whether output to file or printer, may be highlighted to help them stand out from their surroundings, which is useful where there are a relatively small number of hit terms within a record.

Of necessity, there can be no highlighting effect if all hit terms occur in fields not shown by the prevailing output format.

The default **Print Highlight** accent mark is a single asterisk [\*] printed before and after each hit term. If you are using a UNIX printer control file, you can specify an alternative string of marking characters and/or escape sequences in that file and override the highlight default. Refer to the 'Print PRINTER' section in this manual for more information regarding printer control files.

The **DEfine [NO] Highlight** orders affect print output only if you use a **Print Highlight** order.

### Syntax

```
Print HIGHLIGHT
```

Print HIGHLIGHT can also be used with the Local, NOW, WAIT, [NO] HOLD, FILE=, PRINTER=, S=, R=, Format=, REVERSE, SORT= and FOCUS modifiers.

### Examples

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <6>   Find trip AND 3rip
S=3      <20>  Find OR tdbS
S=4      <4>   F AND telex
```

```
Print S=2 HIGHLIGHT Format=content SORT=rname
```

outputs only the contents of the **content** field from search number two, with highlighting of hit terms, and sorted by the contents of the **rname** field.

#### Example 2:

```
Print HIGHLIGHT S=3 R=1,2,3 PRINTER=BOLD
```

prints the first three records in search result set number three with highlighting of hit terms as specified in the printer control file called 'bold.prn'.

## Print List

### Description

This order outputs the current session search history, up to the system limit of 255 search results.

### Syntax

```
Print List
```

**Print List** can also be used with the **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <3>   Find white rabbit AND dormouse
S=3      <1>   Find AND hare
Print List
```

prints the session search history to date.

#### Example 2:

```
S=1      <475> BAsE Alice
S=2      <3>   Find white rabbit AND dormouse
S=3      <1>   Find AND hare
S=4      <1>   Find S=2 AND treacle
Print List FILE=history.lis NOW
```

outputs the current search history to a file called 'history.lis'.

## Print Local (TRIPclassic only)

### Description

This order sends a screen capture of the currently-active window (such as **History**, **Show**, **SStatus**, **Help** or **Display**) to a local printing device.

A **Print Local** order for a **Show** window will, when used without modifiers, output only the currently active record when the order is given. It can, however be used with the same modifiers as a **Show** order for formatting, sorting, record numbers and so on. For details on these modifiers, refer to the sections on 'Show' in this guide.

When used with the **History**, **SStatus** or **Display** windows, **Print Local** will output the complete text or list behind the current window, regardless of what portion of that text or list was displayed when the order was given. **Print Local** takes no modifiers when used with these windows.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

```
Print Local
```

Print Local can also be used with the S=, R=, Format=, REVerse, SORT=, FOCUS, Hlghlight, BAsE, SForm, Procedure, Group, USer, EForm, Format and BAsE= modifiers.

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <21>  Find dr#
Show
Print Local
```

This series opens the demonstration database **Corr**, finds all terms beginning with the letters 'dr', displays the first screen of results and prints the record currently on display, that is, record number forty-eight.

*Example 2:*

```
<Gold><H>
Print Local
```

This series returns the viewer to the history window with **<Gold><H>**, after which the **Print** order will output only the search history rather than the records of the search result.

*Example 3:*

```
S=1      <99>  BAsE corr
S=2      <21>  Find dr#
Print Local R=2 TO 6 SORT=rname DESCending
```

outputs record numbers two through six of the most recent search result, sorted in descending order by **rname**. **R** (**R**ecord number) refers here to the number of the record *within the search result*, not the database record number—in this case, search result record number two is actually database record number fifty-three. Any attempt to output an 'R' value greater than the number of hits in the search result (twelve, in this instance) will result in an error message.



## Print MEasure

### Description

This order prints statistical measures generated for **NUmber** and **INteger** fields.

### Syntax

```
Print MEasure field
```

where *field* is the name of the field to be analyzed. **Print MEasure** can be used with the **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=** and **S=** modifiers.

### Examples

*Example 1:*

```
S=1      <24>  BASe carroll  
S=2      <8>   Find earth or sky or stars  
Print MEasure S=2 chaptnr
```

opens the **Carroll** database, finds all the records which contain the terms 'earth', 'sky' or 'stars' and prints a statistical analysis by the **INteger** field **chaptnr**.



## Print PART SORT

### Description

**PART SORT** is used when sorting records containing part records. If you wish to sort only on head fields, sorting is executed as for records without parts. If you wish to sort on part fields as well, there are two options, *general* and *part sorting*.

#### General SORT

This method sorts records according to field contents, and outputs them in the order specified. Each head record is listed with its parts in the default output format.

#### PART SORT

This option sorts records by *entity* (head plus part record), according to the content of the part fields of each. The entities are sorted within the search set and output as separate units. **Print PART SORT** outputs each part as if it were an individual record in this way, each part preceded by its head record.

Refer to Chapter Eight, 'Head and Part Records' of the *TRIPclassic User Guide* for more information and examples on sorting part records.

### Syntax

```
Print PART SORT={field [DESCending]
[,field [DESCending][,...]]|FREquency} [[NO]MERGe]
```

where *field* is the name of a part field on which to sort. **PART SORT=** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=**, **S=**, **R=**, **Format=**, **FOcus** and **Hlghlight** modifiers.

### Examples

The following examples are for use with the demonstration database **Carroll**.

#### Example 1:

```
S=1      <24>  BAsE carroll
S=2      <6>   Find R=1 TO 3, FROm 22
Print SORT=book
```

prints the records of the latest search result sorted according to the contents of the **book** field, and lists them with all parts in the default output format.

#### Example 2:

```
Print PART SORT=speaker
```

prints the records sorted by entity, according to the content of the **speaker** field in the part records of each.

#### Example 3:

```
Print S=0 PART SORT=speaker  Format=speaker,person
```

outputs the records of the most recent search result sorted by entity, according to the content of the **speaker** field in the part records of each. The run-time output format specified prints only the contents of the **speaker** and **person** fields.

## Print PRINTER

*Note:*

*For UNIX only*

### Description

By default, **Print** output is sent to the printer queue defined by the logical name `TDBS_PRINT` (UNIX only). Output may, however, be redirected using printer control files, either temporarily (at run-time) using **Print PRINTER** or for the duration of the session with **Define PRINTER**.

### Syntax

```
Print PRINTER=printcontrolfile
```

where *printcontrolfile* is a printer control file. This command can be used with any **Print** modifier or qualifier except **Local**.

### Examples

*Example 1:*

```
Print PRINTER=LPT1
```

sends the output to the printer queue defined in the printer control file called `TDBS_PRC/ LPT1.PRN` (UNIX only), when the TRIP session is ended.

*Example 2:*

```
S=1      <475> BAsE Alice
```

```
Print NO HOLD Format=speaker R=20 PRINTER=ptr3
```

immediately prints the contents of the **speaker** field in record number twenty from demonstration database **Alice**. Output is sent to a printer queue defined in the printer control file named `TDBS_PRC/PTR3.PRN` (UNIX only).

## Print PProcedure

### Description

This order prints a list of all the procedures available to the user who issues this order.

For ordinary users, this list comprises all private procedures created by the user, all public procedures, and all procedures created by the user manager for the group to which the user belongs. The system manager may request information regarding all procedures in the system.

### Syntax

```
Print PProcedure [R={ [group.]procedure|ALL}]
```

where *procedure* is the name of the procedure for which information is to be printed, and *group* is the name of the group to which the procedure is available. **Print PProcedure** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
Print PProcedure
```

prints a list of all procedures available to the user making the request.

*Example 2:*

If you save these two CCL instructions as a procedure called 'myproc':

```
S=1      <475> BAsE Alice  
S=2      <16>  Find rabbit hole
```

you can then print the name of that procedure and that of its creator (you) with:

```
Print PProcedure r=myproc
```

*Example 3:*

```
Print PProcedure r=public.macroread
```

allows all members of group 'Public' to print the name of the procedure called 'Macroread' and any existing comments.

*Example 4:*

```
Print PProcedure R=ALL FILE=all_procs.lis
```

allows the user 'System' to print a list of all procedures available in the system to the file named 'All\_procs.lis'.

## Print – Record

### Description

The numbers used in the **Print R=** command refer to *internal record numbers* native to that search set, rather than *database record numbers*. Any record numbers given are relative to, and refer only to the search result set, not the entire database. For example, **R=12** indicates the twelfth record in the search result set rather than record number twelve in the database.

To specify records to be printed by database record number (rather than by the internal record numbers discussed above), add the **BASe=** modifier.

Ranges of record numbers can be specified using (**TO** and **FRom**). Ranges and values are separated by commas.

### Syntax

```
Print R=range [BASe=database]
```

where *range* is the database record number or range of record numbers to be printed, and *database* is the name of the database from which records are to be printed. **Print R=** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=**, **TForm=**, **S=**, **Format=**, **REVerse**, **SORT=**, **FOcus**, **Highlight**, **BASe**, **SForm**, **PRocedure**, **GRoup**, **USer**, **EForm** and **Format** modifiers.

### Examples

#### Example 1:

```
S=1      <99>  BASe corr
Print R=5,3,1 Format=2
```

prints the first, third and fifth records of the most recent search result, using the pre-defined format named '2'. The records are printed in the order in which they were added to the database.

#### Example 2:

```
Print R=1,3 TO 5 SORT=rname FILE=names.fil
```

prints the first, third, fourth, and fifth records of the most recent search result to a file called 'Names.fil'. The records are sorted by the content of the **rname** field.

#### Example 3:

```
S=2      <21>  Find R=55 TO 75
S=3      <35>  Find R=TO 35
S=4      <3>   Find R=38,41,42
Print S=2 R=FRom 10 Format=rname,sname
```

These three search requests find the record numbers indicated, after which the print request returns to search result number two (containing twenty-one records, numbered fifty-five to seventy-five inclusive) and outputs only the **rname** and **sname** fields for the last eleven records in the search set.

#### Example 4:

```
Print BASe=corr R=1 TO 6 NO HOLD
```

prints the first six records of demonstration database **Corr** immediately, in the order in which they were added to the database.

## Print REVerse

### Description

If no sorting is defined during a search session, the system default causes records to be printed or shown in order by record number, that is, the order in which they were added to the database. It is often useful to print the records that were added to the database most recently first.

If the modifier **REVerse** is included, the records are printed in reverse order by record number. The **REVerse** and **SORT** modifiers cannot be used in the same **Show** order. Use the **DESCending** modifier instead to reverse the order of a sorted output.

To redefine the output order for the remainder of the search session, use **DEfine REVerse**—the records most recently added to the database (having the highest record numbers) will always be presented first. To return to the system default of ascending record number order, use **DEfine NO REVerse**.

### Syntax

```
Print [NO] REVerse
```

Print REVerse can also be used with the Local, NOW, WAIT, [NO] HOLD, FILE=, PRINTER=, S=, R=, Format=, Hlghlight and FOCUS modifiers.

### Examples

These examples are intended to be worked sequentially.

#### Example 1:

```
S=1      <99>  BAsE corr
Print REVerse FILE=record.fil
```

prints the records of the most recent search result in reverse order to a file called 'record.fil'.

#### Example 2:

```
S=2      <9>   Find trip
S=3      <7>   Find AND tdbS
S=4      <5>   Find AND 3rip
Print S=3 R=3 TO 7 Format=1 REVerse
```

prints the third through seventh records contained in search result number three in reverse order, using the pre-defined format called '1'.

## Print – Search Result

### Description

If a search result set (**S**) is specified, only the results of that search are printed. Only one search result set may be referenced in a **Print** order at a time.

### Syntax

```
Print [S=n]
```

where *n* is the value of the search result number to be printed. This command can be used with any other print modifier except **List**, **BASe**, **SForm**, **PRocedure**, **GRoup**, **USer**, **EForm**, **Format** and the **Display** *thesaurus modifiers*.

### Examples

#### Example 1:

```
S=1      <475> BASe Alice
S=2      <54>  Find red and queen
S=3      <1>   Find and king
Print S=2
```

prints search result number two. The records are printed in the order in which they were added to the database.

#### Example 2:

```
S=1      <99>  BASe corr
S=2      <5>   Find telex
S=3      <2>   Find and 3rip
Print S=2 Format=2 SORT=rname DESCending FILE=telex.fil
```

outputs the results of search number two using the pre-defined format '2' to a file called 'telex.fil'. The records are sorted in descending order by the contents of the **rname** field.

#### Example 3:

```
S=1      <99>  BASe corr
S=2      <60>  Find mats
Print REVerse FOCUS
```

prints the records located by the latest search result, in focus and in reverse order.

## Print SForm

### Description

This order prints a list of all the search forms available on the TRIP system. The addition of the **R=** construct limits the output to information regarding the specified search form.

### Syntax

```
Print SForm [R=searchform]
```

where *searchform* is the name of the search form to be displayed. **Print SForm** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
Print SForm
```

prints a list of all of the search forms available in the TRIP system.

*Example 2:*

```
Print SForm R=alice_demo FILE=alicedemo.fil
```

outputs information on the search form 'Alice\_demo' to a file called 'alicedemo.fil'.

## Print SORT

### Description

**SORT** is used in **Print** and **Show** orders to arrange output in a predetermined order. Any field type except **STring** may be included in a **SORT** specification, with multiple sort fields separated by commas. The field names and types available for any particular database can be seen in the **STatus** list for that database.

Records where one or more of the sort target fields are empty will appear after all those for which the field has content, sorted in ascending record number order.

The **DESCending** modifier causes **Print** and **Show** output to be arranged alphanumerically, in descending order according to field content.

Normally, records originating from more than one database are sorted *within* the hit record set for their database, each set being sorted individually. The sorted sets are then output in the order in which the databases were opened. Including the **MERGE** qualifier in the **SORT** order intermingles or merges the hit records of all databases into a single set and sorts them together. Use **Define MERGE** to make merged sorted outputs the default within a session; use **Define NO MERGE** to cancel merged sorting.

If the records in the search result were retrieved using the **FUZZ** command, it is useful to view them ranked in order of relevance. To do this, use **SORT=Frequency**. Using **SOR=FRE** on a non-**FUZZ** search result orders the records by number of occurrences.

Sorting on records which contain part records is complex. Refer to the section 'Output and Sorting' in Chapter Eight of the *TRIPclassic User Guide*, 'Head and Part Records' for further information.

### Syntax

```
Print SORT={field [DESCending]
            [,field [DESCending][,...]]|Frequency} [[NO]MERGe]
```

where *field* is the name of the field to be sorted on. **Print SORT=** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=**, **PRINTER=**, **S=**, **R=**, **Format=**, **FOCUS**, **Highlight**, **Display** and **TForm=** modifiers.

### Examples

The following examples were taken from the **Alice** demonstration database provided with TRIP.

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <2>   Find miles
Print SORT=chaptnr
```

prints the results of the most recent search in the default format, sorted with the record with the lowest value of **chaptnr** first and the remainder in *ascending* order by **chaptnr**. Records which have identical values for **chaptnr** appear in the order in which they were added to the database.



*Example 2:*

```
S=3      <4>   Find black
S=4      <2>   Find and kitten
Print S=3 SORT=person, chaptnr
```

prints the records of the third search result in the default format, sorted first in standard alphabetical order according to the contents of the **PHrase** field **person**, and in numerical ascending order by the content of **chaptnr**. Records with identical **chaptnr** value(s) appear sorted by the content of the **person** field, and vice versa.

*Example 3:*

```
S=5      <16>  Find rabbit hole
Print SORT=chaptnr DESCending, person, speaker  DESCending
Format=chaptnr, person, speaker
```

prints only the **chaptnr**, **person** and **speaker** fields of the most recent search result (S=5), sorted first in descending numerical order on the contents of **chaptnr**, then in standard or ascending alphabetical order by **person** and descending alphabetical order by **speaker**.

Records with identical values for **chaptnr** appear sorted alphabetically by the content of the **person** field; records containing the same values for both **chaptnr** and **person** are sorted in reverse alphabetical order by **speaker**.

*Example 4:*

This example is taken from a cluster containing the **Alice** and **Carroll** databases.

```
S=1      <499> BAsE all=alice, carroll
S=2      <33>  Find gryphon
Print Format=person SORT=person DESCending  MERGe
FILE=person.dat
```

prints the contents of the **person** field from the most recent search (S=2) to a file called 'person.dat', with all hit records sorted by the content of **person** in descending order.

## Print TForm

### Description

This order is used to output records in TRIP's predefined system file format *TripFormat*, that creates a file in the format **TForm**. It is also useful in updating databases. Records stored in a **TForm** file can be edited if necessary, then loaded into another database. When the file name given in the order has no extension, the default extension will be '.TFO'.

The addition of the **R** or *record identifier* modifier (as opposed to **R=**, the output record number modifier) instructs TRIP to include the record identifier in each record sent to the **TForm** file. The record identifier is the record name if available, or the record number if it is not.

### Syntax

```
Print [{NOW|WAIT|[NO]HOLD}] TForm=[{'|'}][path]filename
[{'|'}] [R]
```

where *path* is the (optional) path to *filename* and *file* is the name of the receiving **TForm** file. **Print TForm=** can also be used with the **NOW**, **WAIT**, **[NO] HOLD**, **S=**, **R=**, **REVERSE** and **SORT=** modifiers.

*Note:*

*File names (with or without path strings included) that contain spaces must be enclosed in single (') or double (") quotes.*

*Notes:*

- *If printing from within TRIPmanager, you MUST use the NOW modifier or nothing will be printed.*
- *PRINT FILE overwrites existing files of the same name with NO WARNINGS.*
- *If the file path is incorrect, or if access rights to the location for the file to be printed are incorrect, a 'file could not be opened' error is appended to the print instruction in the PRxxxxx.log file.*
- *If no path is specified, the file will be printed to the current process' directory. For example, on Windows 7:*
  - *for the 'TRIP classic' start menu shortcut, this will be the <TRIPinstallation>\bin directory*
  - *for a TRIPclassic session started in a 'cmd' window, this will be the directory the session was started in*
  - *for TRIPmanager this will be the \windows\system32 directory*

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <6>   Find royalty
S=3      <6>   Find AND 3rip
S=4      <5>   Find S=3 NOT inrip
Print S=3 TForm=transfer
```

writes the records of search result three into a text file named 'transfer', using **TripFormat**. TRIP will automatically append the extension '.tfo' to a file such as this, where no extension has been provided.

*Example 2:*

```
S=5      <99>  BAsE corr
Print R TForm=transfer.trp
```

copies all records in database **Corr** into a **TForm** text file called 'transfer.trp', and inserts a record identifier (in this case, the record number) in each.

*Example 3:*

```
S=6      <37>  Find rcomp=paralog
Print TForm=paralog.tfo SORT=rname DESCending      NOW
```

locates all records in **Corr** that contain the name 'paralog' in the **rcomp** field, sorts them in descending order by **rname**, and immediately prints them to the file called 'paralog.tfo'.



## Print – Timing

### Description

There are four **Print** modifiers in TRIP which affect the immediacy with which the print job is started: **HOLD**, **NO HOLD**, **NOW** and **WAIT**. Only one of these may be included in a CCL order.

#### **HOLD**

**HOLD** is the default, and so is the modifier used by TRIP when no modifier is given in a **Print** order. It accumulates all print requests made during a session and submits them as a batch job when the TRIP session is ended.

With **HOLD**, all **Print** requests being held for later execution can be listed using the order **Print?** Where necessary, a print request can be deleted using the command **DElete Print=*n***, where *n* is the print request number to be deleted. **DElete Print=ALL** deletes all deferred print requests.

The search which generated the result set to be printed cannot be deleted, to prevent the print request from aborting. If searches are to be deleted, one of the other **Print** modifiers should be used.

**DEfine NO HOLD** cancels the system print default.

#### **NO HOLD**

**NO HOLD** submits print requests as a batch job to the batch queue as soon as the order is given, rather than holding them until the end of the session.

#### **NOW**

**NOW** and **WAIT** are different in that the print job is run as a sub-process in the background, rather than as a separate process. The print job now cannot be delayed by other jobs in the queue.

**NOW** and **WAIT** cannot be set as defaults and must be used explicitly in the **Print** order.

**NOW** runs a sub-process in parallel to the TRIP session, and so allows searching to continue. However, if the host session is terminated before the sub-process completes, the print job will fail, and if the sub-process is still running when the TRIP session is terminated, TRIP issues a warning to this effect.

#### **WAIT**

**WAIT** runs a sub-process and blocks all further activity until the print job has completed. This is useful in procedures and macros where the CCL orders following **Print** depend on the completion of the print process.

### Syntax

```
Print [{NOW|WAIT| [NO]HOLD}]
```

**Print [NO] HOLD/WAIT/NOW** can be used with any other print modifier or qualifier except **Local**.

## Examples

Consider these examples from **Alice** as having been given consecutively:

### Example 1:

```
S=1      <475> BAsE Alice
Print R=16
Print?
```

This series opens database **Alice**, locates database record number 16 and stores the **Print** order 'Print S=1 R=16' as print request number one.

### Example 2:

```
DEfine NO HOLD
Print R=128
Print?
```

Building on the previous search history, these orders set **NO HOLD** to be the default, locate database record number 128 and submit the second **Print** order to the print queue as 'Print S=1 R=128', and show the first **Print** order still holding.

### Example 3:

```
Print NOW R=25 TO 30
```

locates database records twenty-five through thirty and begins the third **Print** order as a sub-process in the background.

### Example 4:

```
Print WAIT R=TO 40
Print?
STOP
```

locates database records one through forty and starts the fourth **Print** order in the foreground, shows the first **Print** order still holding and ends the TRIP session with **STOP**.

The first **Print** order is now submitted to the print queue.

## Print TRace

### Description

This command prints the history of a particular search or a selection of searches. If used without the **S=** modifier (which specifies single search result numbers or ranges of numbers), TRIP will print the history of the most recent search performed.

### Syntax

```
Print TRace
```

**Print TRace** can also be used with the **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

*Example 1:*

```
Print TRace
```

prints the history of the most recent search performed.

*Example 2:*

```
Print TRace S=1
```

prints the history of the first search performed.

*Example 3:*

```
Print TRace S=FRom 4
```

prints the history of all but search result numbers one through four.

## Print USer

### Description

This order prints read and write access information for users of the TRIP system. Users are presented in alphabetical order by TRIP user name, while the databases to which each has access are listed chronologically by access date.

### Syntax

```
Print USer [R={user|ALL}]
```

where *user* is the TRIP user name for which information is being requested. **Print USer** can also be used with the **Local**, **NOW**, **WAIT**, **[NO] HOLD**, **FILE=** and **PRINTER=** modifiers.

### Examples

#### Example 1:

```
Print USer
```

prints read and write access information for all users created by the user manager requesting the information.

#### Example 2:

```
Print USer R=george
```

prints access information for TRIP user 'George'. If user 'George' is the user needing this information and he is not a user manager, he will only be able to print his own access information. If the requester is the user SYSTEM or the user manager who owns the user whose status is being requested, the user's read and write privileges will be printed.

#### Example 3:

```
Print USer R=ALL PRINTER=lpt1
```

prints access privilege information for all TRIP system users using the printer control file called 'LPT1.PRN'. This order is available only to the TRIP system manager.

## RENumber

### Description

A **RENumber** order compresses the sequence of search set numbers until there is no hole in the sequence.

**RENumber** may be used to make room for more searches during a session by:

Delete unnecessary searches using **DElete S=n**

Renumber the remaining search numbers using **RENumber**.

*Note:*

*The renumbering will only work if the previously deleted sets are not referred to in any of the remaining sets.*

### Syntax

To renumber the search history.

```
RENumber
```

### Examples

*Example 1:*

```
S=1      <475> BAsE Alice
```

```
S=3      <11>  Find TExt=unicorn
```

**RENumber**

will result in the following search history.

```
S=1      <475> BAsE Alice
```

```
S=2      <11>  Find TExt=unicorn
```



## REVeal (TRIPclassic only)

### Description

*This command is chiefly for use in procedures, acting to restore output of CCL orders to the screen after a **HIDE** command. Use **HIDE** to prevent CCL commands from echoing to the screen. Neither command accepts arguments.*

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

REVeal



## Run

### Description

Search results from a TRIP search session can be saved for future use with the **SAve** command. Saved searches, macros and procedures can all be executed either by using their names as a command, or by using the **Run** order.

When a saved search or procedure is run, the results or search orders are shown to screen in the same way as during any other search.

### Syntax

```
[Run] procedure [argument[,argument[,...]]]
```

where *procedure* is the name of the procedure to be executed, and *argument* is a string that will be passed to that procedure as an argument.

### Examples

#### Example 1:

If you save the first two CCL instructions shown below as a procedure called 'myproc' as illustrated:

```
S=1      <475> BAsE Alice
S=2      <16>  Find rabbit hole
SAve myproc
```

you can then reactivate those orders on the CCL command line with:

```
Run myproc
```

#### Example 2:

If you execute the first three CCL instructions shown below:

```
S=1      <99>  BAsE corr
S=2      <56>  Find PHrase=mats
S=3      <10>  Find NOT g
SAve mysearch
```

and save them under the name 'mysearch' as illustrated, you can then run them as a procedure on the command line, as illustrated below:

```
mysearch
```

#### Note:

*When using procedures in an application program (client or server based) with TRIPsystem versions prior to TRIPsystem version 3.4, it was necessary to take care of all post-processing after executing a TRIP procedure using the CCL command: RUN myproc.*

Now, if such a procedure contains only search related commands (e.g. FIND, BASE, DEFINE, etc.) TRIPsystem will carry out all the commands as expected, if executed using the CCL command FIND SAVE myproc.

## SAve

### Description

The **SAve** order is used to save search orders, *not* search results, for use in a later search session. It must contain the name you wish to give the saved routine, and may also contain the number or numbers of the search result(s) to be saved. When given without search result numbers, the **SAve** order saves the most recent search.

Ranges of searches are defined in the usual way using **TO**, **FFrom** and the comma character.

When a saved search routine is run, it often produces a different search result from the one it produced when it was saved. When a search order is saved, only the orders it depends on for the final result are saved.

Search orders are actually saved in a private procedure, where they may be further edited or manipulated. Refer to Chapter Eleven, 'User Procedures' of the *TRIPclassic User Guide* for more information regarding procedures.

A search session can be effectively 'frozen' and your current search orders saved if you exit TRIP using the command **STOP SAve**. Among other things, both the search orders and the search results are saved to the **SIF** (Session Index File), which allows you to begin searching where you left off when you begin a new TRIP session. When you enter CCL, the search orders and results are reported in the history window as they appeared when you last left TRIP. See the 'STOP' section for details.

It is possible to exclude the initial **BASe** order from a set of saved search commands using **DEfine SAve NO BASe**. See the section in this guide entitled 'DEfine SAve BASe' for further information.

Saved searches may be deleted using a **DElete** order. Refer to the sections on 'DElete' in this manual for details.

### Syntax

```
SAve [S=range] savedsearch
```

where *range* is one or more search result numbers to be saved and *savedsearch* is the name of the procedure to be created.

### Examples

#### Example 1:

```
S=1      <499> BASe all=alice,carroll
S=2      <21> Find haigha OR servants
S=3      <456> Find white rabbit OR Alice
S=4      <2> Find rabbits
S=5      <457> Find S=2 OR S=3
SAve S=TO 2,5 mysearch
```

this series:

1. opens the cluster database called **All** (S=1),
2. finds the records containing 'haigha' or 'servants' (S=2),

3. finds the records containing 'white rabbit' or 'Alice' (S=3),
4. finds all the records containing 'rabbits' (S=4),
5. finds all the records of searches two and three (S=5),
6. and saves all the searches of the current session *except* number four in a private procedure called 'Mysearch'.



## Select (TRIPclassic only)

### Description

The **SElect** command (or <**Select**> key) moves the cursor from the CCL command window to the **Show** window during a **Show** order. Use the <**Next**>, <**Prev**>, <**↑ Arrow**> and <**↓ Arrow**> keys to move about the display, and <**Space**>, <**Enter**> or <**Return**> to move back to the command window. The record the cursor was positioned beside or in then becomes the current record.

The **SElect** command can also be used to cut a piece of text from the **Show** window for storage in a file. First, specify a 'cut' file using a **DEfine CUT=filename** order (see the 'DEfine CUT' section in this manual for more information). Next, position the cursor as described above, then mark the beginning of the cut area by pressing <**Select**>. Move the cursor to the end of the area you wish to select and press <**Cut**>. The excerpt is then printed to the 'cut' file that you defined previously.

You may also use this command to choose a subset of the records of a search result to form a new search result. If you use **SElect R** after a **Show** request, the cursor moves to the **Show** window. Use the <**Next**>, <**Prev**>, <**↑ Arrow**> and <**↓ Arrow**> keys to move about the display, and <**Select**> to pick the record(s) at the cursor. Pressing <**Enter**> returns the cursor to the command window and forms a new search result containing only the records you have selected.

You may also choose terms and build a restricted search result from a **Display** list in this manner.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

```
SElect [R]
```

### Examples

*Example 1:*

```
S=1      <99>  BAsE corr
S=2      <60>  Find mats
Show
SE R
```

In showing the sixty records produced by search result number two, the user decides that he or she would like to limit searching to record numbers one, three and eight. After selecting them, TRIP builds this search result:

```
S=3      <3>   Find (corr.R=1, 3, 8) AND.R S=2
```

which contains only the selected records.

**Example 2:**

```
S=1      <475> BAsE Alice
Display hat#
```

This produces the following **Display** list:

(Display hat#, 7 terms)

```
T=1      <1>   HAT
T=2      <1>   HATCHING
T=3      <3>   HATE
T=4      <1>   HATED
T=5      <6>   HATTA
T=6      <28>  HATTER
T=7      <1>   HATTERS
```

If you then give the request

```
SE
```

select 'Hatta', 'Hatter' and 'Hatters' from the list and press **<Enter>**, TRIP will execute a **Find** order to locate all records containing these three terms:

```
S=5      <35> Find ("HATTA" OR "HATTER" OR
                  "HATTERS")
```

## SForm

### Description

This command enables the user to enter a search form from CCL or the menu of another search form.

*Note:*

*This command is valid in TRIPclassic only.*

### Syntax

```
SForm searchform
```

where *searchform* is the name of the search form to call. **SForm** can also be used as a modifier in **Show** and **Print** orders to obtain information about the search forms available in the TRIP system. Refer to the sections entitled 'Show SForm' and 'Print SForm' in this guide for further information.

### Examples

*Example 1:*

```
SForm alice_demo
```

opens the search form named 'Alice\_demo' for the demonstration database **Alice**.

## Show

### Description

There are three types of **Show** orders: those that show the results of searches, those that provide information about the system and those that give accounting data. When **Show** is used to display the records of a search result on screen, the output format determines what portion of each record is presented.

Use the **More** and **Back** commands, and the <Next>, <Prev>, <↑ Arrow> and <↓ Arrow> keys to scroll through the output. <Next> (or <Gold><N>) places the cursor at the beginning of the next record, <Top> (or <Gold><T>) positions the cursor at the beginning of the current record and <Gold><H> returns you to the search history window.

### Show – Section Index

Show orders are discussed under the following headings:

Table 25 – List of SHow orders

Command	Subject
Show	A Simple Order
Show	ACcess
Show	BASe
Show	COST
Show	EForm
Show	FOcus
Show	Format
Show	GRoup
Show	Highlight
Show	PART SORT
Show	PRocedure
Show	Record
Show	REVerse
Show	Search Result
Show	SForm
Show	SORT
Show	USer



## Syntax

To show the contents of a search result:

```
Show { [[FROm]R] | [{S=n|BAsE=database}] [R=range] }
      [Format={format|{field|fieldtype|view}
      [, {field|fieldtype|view} [, ...]]}]
      [{[NO]REVerse| [PART]SORT={field[DESCending]
      [, field[DESCending] [, ...]] | FRequency} [ [NO]MERGe} ] ]
      [[NO]FOcus] [[NO]HIGHLIGHT]
```

To show information regarding databases, forms, etc.:

```
Show { {BAsE[LISt] | [BAsE]ACcess|PRocedure|
      GRoup|USer} [R={item|ALL}] | SForm [R=item] |
      {EForm|Format} {BAsE=database|R=ALL} ] }
```

To show the accumulated cost for records shown or printed:

```
Show COST
```

where:

- *n* is a search result number
- *range* represents a record number, list of record numbers or range of record numbers
- *format* is the name of an output format to be used
- *field*, *fieldtype* and *view* are fields, field types and views whose contents are to be shown
- *item* is the name of a database, search form, procedure, user or group. *Item* may be truncated.
- *database* is the name of a database for which information is to be shown, or to which record numbers refer.

## Show – A Simple Order

### Description

If a **Show** command is given alone, the results of the most recent search (S=0) will be shown using the default output format for the open database. The records will not be sorted or focused, but all hit terms will be highlighted.

### Syntax

Show

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <1>   Find twiddle
Show
```

shows the result of the most recent search (S=2) using the currently defined output format, with the hit terms highlighted. If more than one record is found, they are shown in the order in which they were added to the database.

#### Note:

*It is also possible to show information from the CONTROL database using truncation:*

```
Show PRocedure R=username.#art#
```

*would show to the user username, all procedures that belong to them with a name matching the truncated string #art# e.g. art, cartesian, tarzan etc. and:*

```
Show BAsE=#rr#
```

*would list all databases with a name matching the truncated string #rr#, e.g. Corr, Carrie, Carrot, etc.*

## Show ACcess

### Description

This order enables database administrators (file managers) and the user called SYSTEM to display access information for users of their databases. Databases are presented in alphabetical order, while users having access are listed chronologically by access date.

The addition of the *database* construct limits the screen output to the specified database only. This order is equivalent to **Show BAsE ACcess**.

### Syntax

```
Show [BAsE] ACcess [R={database|ALL}]
```

where *database* is the database whose status is requested.

### Examples

#### Example 1:

```
Show ACcess
```

```
Show BAsE ACcess
```

Both of these orders give the read and write privileges of each user and user group for all of the databases that the database administrator has created. The databases are arranged alphabetically, and the user and user groups listed in chronological order by date of creation.

#### Example 2:

```
Show ACcess R=Alice
```

```
Show BAsE ACcess R=Alice
```

Both orders show the read and write access privileges for all users and groups with access to the demonstration database **Alice**. Only the creator of the database in question may request access information for that database. The user called SYSTEM may request access information for any database.

#### Example 3:

```
Show ACcess R=ALL
```

```
Show BAsE ACcess R=ALL
```

Both orders display the access rights for all users and user groups of all databases on the TRIP system, and are available only to user SYSTEM.

## Show BASe

### Description

This order allows users to view the status listings of all of the databases to which he or she has read access. The addition of the *database* construct limits the screen output to the specified database only.

**Show BASe** without modifiers shows complete status listings for all databases that the user has created. **Show BASe List** shows a less-detailed list of databases to which the user has read-access. **Show BASe R=database** shows a status listing for only the specified database. **Show BASe=database** shows all of the records of the specified database.

### Syntax

To show status screens for or information about databases:

```
Show BASe [List] [R={database|ALL}]
```

where *database* is the name of the database whose status is to be shown.

To show all of the records of a database:

```
Show BASe=database
```

where *database* is the name of the database from which records are to be shown. **Show BASe=** can also be used with the **S=**, **R=**, **Format=**, **REVerse**, **SORt=**, **FOcus** and **Highlight** modifiers.

### Examples

*Example 1:*

```
Show BASe
```

presents the status lists for all databases owned by the database administrator making the request, in alphabetical order.

*Example 2:*

```
Show BASe R=corr
```

shows the status for the demonstration database **Corr**. This order is equivalent to **Status**.

*Example 3:*

```
Show BASe R=ALL
```

shows the status screens for all databases on the TRIP system, and is available only to the user called SYSTEM.

*Example 4:*

```
Show BASe LIst
```

shows a list of all databases to which the user has read-access, with their owners and descriptions.

*Example 5:*

```
Show BASe LIst R=corr
```

shows the name, owner and description of the demonstration database **Corr**.

*Example 6:*

```
Show BASe LIst R=ALL
```

shows the name, owner and description of every database on the TRIP system, and is available only to the user called SYSTEM.

*Example 7:*

```
Show BASe=corr
```

shows all of the records of demonstration database **Corr**, in the order in which they were added to the database.

*Example 8:*

```
Show BASe=corr R=1 TO 6
```

shows the first six records of demonstration database **Corr**, in the order in which they were added to the database.



## Show COST

### Description

This **Show** modifier gives information on accumulated costs for records shown or printed in the current TRIP session, if these have been defined for the database in question. It also provides information regarding the time elapsed and other user-specific statistics.

### Syntax

```
Show COST
```

### Examples

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <1>   Find twiddle
Show
S=3      <2>   Find cook and cauldron
Show
Next
S=4      <1>   Find mome
Show
Show COST
```

presents accounting information for the entire TRIP search session to date, in this case stating that you have viewed a total of four records from database **Alice**.

## Show EForm

### Description

This order shows a list of all the entry forms accessible to the user. The addition of the *database* construct limits the output only to forms of the database given.

A user can create an entry form for any database to which he or she has write access, whether or not he or she owns that database. To view these entry forms, use the **Show EForm BAsE=** command illustrated below.

### Syntax

```
Show EForm [{BAsE=database|R=ALL}]
```

where *database* is the name of the database for which data entry form information is to be provided.

### Examples

*Example 1:*

```
Show EForm
```

displays a list of the entry forms available for all databases owned by the user making the **Show** request, in alphabetical order by database name.

*Example 2:*

```
Show EForm BAsE=Alice
```

prints entry form information for demonstration database **Alice**.

## Show FOCUS

### Description

**FOCUS** allows the presentation of only the paragraph of a **TEXT** field or subfield of a **PHRASE** field that contains a hit. Both field name and record number of each hit record are shown in the output. If all occurrences of the term to be focused exist in fields not included in the output format, the output result will be empty.

Focused outputs can be made the default by giving the order:

```
DEfine FOCUS
```

Use the **Expand** request (or **<Gold><X>**) to view the remainder of a **FOCUSED** record. Once a record is expanded, only that record is available for browsing. The **<↑ Arrow>** and **<↓ Arrow>** keys and the commands **More** and **Back** will operate only within the expanded record, and no other records can be examined unless a new **Show** order or the **Continue** command is given.

To return to the focus point after an **EXpand** request, use the order **Continue Show**.

To cancel the focusing effect temporarily, use **Show NO FOCUS**. To cancel focusing for the remainder of the session, use **DEfine NO FOCUS**.

### Syntax

```
Show [NO] FOCUS
```

**Show FOCUS** can also be used with the **S=**, **R=**, **Format=**, **REVerse**, **SORt=**, **BASE=** and **Highlight** modifiers.

### Examples

These examples use the demonstration database **Alice**, and should be worked consecutively.

#### Example 1:

```
S=1      <99>  BASE Alice
S=2      <28>  Find mad hatter
Show FOCUS
```

shows the most recent search result in focus in the default format.

#### Example 2:

```
DEfine Focus
S=3      <21>  Find S=2 AND Alice
Show R=3 TO 7
```

sets the default for all subsequent **Show** orders to **FOCUS**, finds all records containing the terms 'mad hatter' and 'Alice' and shows the third through seventh records of search result number three in focus, in the default format.

#### Example 3:

```
S=3      <21>  Find S=3 AND hare
Show NO FOCUS
```

finds all records containing the terms 'mad hatter', 'Alice' and 'hare', and shows them without focus.



## Show Format

### Description

A database may have several standard or pre-defined output formats, one of which may have been designated the default format for use during searching. Both pre-defined or run-time output formats can be used with **Show Format** to present only selected information from within a search result.

Pre-defined 'named' formats can be created by any user with read access to the database in question, and those available for any database can be seen in the **Status** list. Use the **Show Format** order to choose a pre-defined format for use with a particular search result, and **Define Format** to change the predefined format to a new one for the remainder of the search session.

Run-time formats are lists of field names, field types and/or views (separated by commas) to be output, and are defined in the **Show** order directly. The available field names and types for any database can be seen in the **Status** list. If a particular run-time combination of field names, types or views is used frequently, it can be defined as the default for the remainder of the search session using a **Define Format** order.

Although the three types of field qualifier, *field*, *fieldtype* and *view*, can be mixed in the same **Show** order, named and run-time formats cannot be mixed in a single order.

The addition of the *database* construct limits the output to formats of the database named.

### Syntax

For a named format:

```
Show Format=format
```

For a run-time format:

```
Show Format={field|fieldtype|view}
[, {field|fieldtype|view}[,...]]
```

**Show Format=** can also be used to show a list of all of the output formats for databases owned by the requesting user, as well as with the **S=**, **R=**, **REVerse**, **SORT=**, **BASE=**, **FOCUS** and **HIGHLIGHT** modifiers.

To display available formats:

```
Show Format [{BASE=database|R=ALL}]
```

where *format* is the name of the desired pre-defined format, *field*, *fieldtype* and *view* are the names of fields, field types and views to be output, and *database* is the name of a database for which the user wishes to see named output format information.

### Examples

The following examples are for use with the demonstration database **Alice**, and are to be worked consecutively.

*Example 1:*

```
S=1      <475> BASE Alice
S=2      <1>   Find twiddle
Show Format=short
```

shows the results of the most recent search (S=2) using a predefined format called 'Short'.

*Example 2:*

```
Show Format=dump
```

shows the contents of *all* the non-empty fields in the database for the most recent search result (S=2). The field contents are preceded by the field name, and the fields themselves are output in a pre-defined sequence according to field type: **PHrase** fields first, then **NUmber**, **INteger**, **DAte**, **TIme** and **TEText**.

The 'dump' format is a pre-defined format supplied with TRIP.

*Example 3:*

```
S=3      <5>   Find red king
S=4      <4>   Find and Alice
Show S=3 R=TO 3 Format=full
```

shows the first three records of search result number three (*not* record numbers one through three) using the format named 'Full'.

*Example 4:*

```
Show Format=chaptnr, person
```

shows the contents of the fields **chaptnr** and **person** from the most recent search (S=4), where these are not empty. The field contents are preceded by the field name, and the fields are output in the order given in the **Show** order (in this case, **chaptnr** appears before **person**).

*Example 5:*

```
Show Format=TEText, INteger
```

shows the contents of all non-empty **TEText** and **INteger** fields in the latest search result (S=4), headed by their field names.

*Example 6:*

```
Show Format=chaptnr, PHrase
```

shows the contents of **chaptnr** and all **PHrase** fields from the most recent search result (S=4), where these are not empty. The fields are output in the order specified in the **Show** request, with the **PHrase** fields appearing in the order in which they were created in the database.

*Example 7:*

```
Show S=2 Format=person, chaptnr
```

shows the records of search result two, in a run-time format that displays only the contents of the fields **person** and **chaptnr**.

*Example 8:*

```
Show Format
```

lists all of the formats for the databases created by the user making the request.

*Example 9:*

```
Show Format BASe=Alice
```

gives a list of all of the predefined formats for the demonstration database **Alice**. This order may be given by any user with read access to the database in question.

*Example 10:*

```
Show Format R=All
```

when entered by the user SYSTEM, displays all of the output formats created for all databases on the system.



## Show GRoup

### Description

The **GRoup** modifier requests information regarding TRIP user groups in the system, and as such is available only to user managers and the user SYSTEM.

### Syntax

```
Show GRoup [R={group|ALL}]
```

where *group* is the name of the group for which membership information will be displayed.

### Examples

#### Example 1:

```
Show GRoup
```

displays a list of group members and their access privileges for all groups created by the user manager giving the order.

#### Example 2:

```
Show GRoup R=sales
```

may be used only by the creator of the group called 'Sales', and presents only the members and member privileges of that group.

#### Example 3:

```
Show GRoup R=public
```

may be used only by SYSTEM (the creator of the group called 'Public'), and presents only the members and member privileges of that group.

#### Example 4:

```
Show GRoup R=ALL
```

may be used only by the user called SYSTEM, and shows membership information for all groups in the system.

## Show Highlight

### Description

Hit terms within output results may be highlighted to help them stand out from their surroundings, which is useful where there is a small number of hit terms within a record.

The default is **Highlight=ALL**, which causes *all* hit terms to be highlighted in **Show** orders. No highlighting will occur if all hit terms occur in fields not shown by the output format currently in use.

To temporarily override the current default for the highlighting of hit terms in a **Show** order, use a **Highlight/NO Highlight** modifier. The **NO Highlight** modifier cancels highlighting, while **Highlight** restores it.

### Syntax

```
Show [NO] Highlight
```

**Show Highlight** can also be used with the **S=**, **R=**, **Format=**, **REVerse**, **SORt=**, **BASe=** and **FOCUS** modifiers.

### Examples

The following examples are for use with the demonstration database **Alice**, and are to be worked consecutively.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <1>   Find twiddle
S=3      <1>   Find tweedle
S=4      <2>   Find OR S=2
S=5      <2>   Find dee
Show S=4
```

shows the results of search number four with highlighting of hit terms.

#### Example 2:

```
S=6      <11> Find TExt=unicorn
Show R=2 TO 5, FRom 9 NO HHighlight FOCUS
```

shows records two through five, and from nine upwards in search result number six, without highlighting and in focus.

## Show PART SORT

### Description

**PART SORT** is used when sorting records containing part records. If you wish to sort only on head fields, sorting is executed as for records without parts. If you wish to sort on part fields as well, there are two options, *general* and *part sorting*.

#### General SORT

This method sorts records according to field contents, and outputs them in the order specified. Each head record is listed with its parts in the default output format.

#### PART SORT

This option sorts records by *entity* (head plus part record), according to the content of the part fields of each. The entities are sorted within the search set, and output as separate units.

Refer to Chapter Eight, 'Head and Part Records' of the *TRIPclassic User Guide* for more information and examples on sorting part records.

### Syntax

```
Show PART SORT={field [DESCending]
                [,field [DESCending][,...]] | FRequency} [[NO]MERGe]
```

where *field* is the name of a field to be sorted on. **Show PART SORT=** can also be used with the **S=**, **R=**, **Format=**, **FOCUS** and **HIGHLIGHT** modifiers.

### Examples

The following examples are to be used with the demonstration database **Carroll**, and should be worked consecutively.

#### Example 1:

```
S=1      <24>  BAsE carroll
S=2      <11>  Find king
S=3      <15>  Find queen
S=4      <9>   Find red
S=5      <5>   Find AND S=2
Show SORT=speaker
```

sorts records according to the collected field contents of the part field **speaker**, and lists them with all parts in the default output format.

#### Example 2:

```
Show PART SORT=speaker Format=book, chaptnr, speaker
```

sorts the records of the most recent search result (S=5) by entity, according to the content of the part record field **speaker** of each. Only the contents of **book**, **chaptnr** and **speaker** are output.

## Show PProcedure

### Description

This order shows a list of all the procedures available to the user who issues this order.

For ordinary users, this list comprises all private procedures created by the user, all public procedures, and all procedures created by the user manager for the group to which the user belongs. The system manager may request information regarding all procedures in the system.

### Syntax

```
Show PProcedure [R={ [group.|username.]procedure|ALL|#}]
```

where *procedure* is the name of the procedure for which information is to be shown, *username* is the name of the user to which the procedure is available and *group* is the name of the group to which the procedure is available.

### Examples

#### Example 1:

```
Show PProcedure
```

displays a list of all procedures available to the user making the request.

#### Example 2:

If you save these two CCL instructions as a procedure called 'myproc':

```
S=1      <475> BAsE Alice  
S=2      <16>  Find rabbit hole  
SAve myproc
```

you can then list the procedure name and that of its creator (you) onscreen with:

```
Show PProcedure R=myproc
```

#### Example 3:

```
Show PProcedure R=public.macroread
```

allows all members of group 'Public' to view the name of the procedure called 'Macroread' and any existing comments.

#### Example 4:

```
Show PProcedure R=ALL
```

allows the system manager to show a list of all procedures available in the system.

#### Example 5:

```
Show PProcedure R=ERIC.ALL
```

Shows the user *Eric* a list of all procedures available to him.

**Example 6:**

```
Show PRocedure R=username. (note the final dot).
```

This command can also be entered as:

```
Show PRocedure R=.
```

or

```
Show PRocedure R=username.#
```

Shows the user *username* a list of all his own procedures.





## Show – Record

### Description

If record numbers are given in a **Show** order, only those records within that search result set are shown.

*Note:*

*The record numbers are relative to, and refer only to, the search result set, not the whole database; i.e. **R=1** implies the first number in the result set, not the first record in the database.*

To specify records to be shown by database record number (rather than by the internal record numbers discussed above), add the **BASe=** modifier.

Ranges of record numbers can be specified using **TO**, **FROm** and/or exact values. The ranges and values are separated by commas.

**Show R** used without record number(s) operates only on the current record. TRIP considers the current record to be the one positioned nearest the bottom of the screen in a **Show** order.

### Syntax

```
Show {[FROm]R|R=range [BASe=database]}
```

where *range* is a record number, record number list or range of record numbers to be examined, and *database* is the name of the database from which records are to be taken. **Show R=** can also be used with the **S=**, **Format=**, **REVerse**, **SORe**, **FOCUS** and **HIGHLIGHT** modifiers.

### Examples

The following examples are for use with the demonstration database **Alice**, and should be worked consecutively.

*Example 1:*

```
S=1      <475> BASe Alice
S=2      <2>   Find dee
S=3      <11>  Find TExt=unicorn
Show Format=speaker
(Position record number 400 at the bottom of the screen)
Show R Format=1
```

shows the active or current record in the current search result (S=3, database record number 400) using the format called '1'.

*Example 2:*

```
Show
Show R=5,3,1 Format=speaker
```

**Show** displays the entire contents of the original **Show** order once again (in this case, **Show Format=speaker**). The **Show R** request shows the contents of the **speaker** field for the fifth, third and first records of the most recent search result, S=3 (*not necessarily record numbers one, three and five*). The records are shown in the order in which they

were added to the database; that is, record number one first, followed by three and five (records 393, 397 and 400 of the database).

*Example 3:*

```
Show R=1,3 TO 5 Format=chaptnr,person
```

shows the contents of the **chaptnr** and **person** fields for the first, third, fourth, and fifth records of the most recent search result (S=3, database numbers 393, 397, 399 and 400). The records are shown in the order in which they were added to the database.

*Example 4:*

```
Show
```

(Position record number 400 at the bottom of the screen)

```
Show FFrom R Format=3
```

makes the previous search set available in its entirety, makes database record number 400 the current record and displays this current record as well as all subsequent records of the latest search result in the format called '3'.

## Show REVerse

### Description

If no sorting is defined during a search session, the system default shows records in order by record number, that is, the order in which the records were added to the database. It is often useful to view the records that were added to the database most recently first.

If the modifier **REVerse** is included in a **Show** order, the records are shown in reverse order by record number. The **REVerse** and **Sort** modifiers cannot be used in the same **Show** order. Use the **DESCending** modifier instead to reverse the order of a sorted output.

To redefine the output order for the remainder of the search session, use **DEfine REVerse**—the records most recently added to the database (having the highest record numbers) will always be presented first. To return to the system default of ascending record number order, use **DEfine NO REVerse**.

### Syntax

```
Show [NO] REVerse
```

**Show REVerse** can also be used with the **S=**, **R=**, **Format=**, **FOcus** and **HIghlight** modifiers.

### Examples

The following examples are for use with the demonstration database **Corr**, and are to be worked consecutively.

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <5>   Find scientific
Show REVerse
```

shows the records of the most recent search result (S=2) in reverse order by record number.

#### Example 2:

```
Show R=1 TO 3 REVerse NO HIghlight
```

shows the first three records of the most recent search result (S=2) in reverse order without hit term highlighting, i.e. record number three first, followed by two and one.

#### Example 3:

```
S=3      <4>   Find AND 3rip
S=4      <2>   Find AND database
Show S=3 REVerse
```

shows the records of search result number three in inverse order by record number.

## Show – Search Result

### Description

If a search result set is specified in a **Show** order, only the results of that search are shown. Only one search result set may be referenced per **Show** order.

### Syntax

```
Show [S=n]
```

where *n* is the value of the search result number to be shown. **Show S=** can also be used with the **R=**, **Format=**, **REVerse**, **SORt=**, **FOcus** and **HighLight** modifiers.

### Examples

These examples are taken from the demonstration database **Alice**, and are to be worked consecutively.

#### Example 1:

```
S=1      <475> BAsE Alice
S=2      <5>   Find red king
S=3      <135> Find OR queen
S=4      <3>   Find AND hare
Show S=2
```

shows search result number two in the currently defined output format. The records are shown in the order in which they were added to the database.

#### Example 2:

```
Show S=3 R=1 TO 5, FRom 130 Format=short FOCus
```

displays the first five records and those numbered 130 or higher in search result number three from the previous search series, using the format called 'Short' and using focus.

## Show SForm

### Description

This order shows a list of all available search forms. The addition of the *searchform* construct limits the output solely to the search form named.

### Syntax

```
Show SForm [R=searchform]
```

where *searchform* is the name of the search form for which information is to be listed.

### Examples

*Example 1:*

```
Show SForm
```

lists all search forms available on the TRIP system.

*Example 2:*

```
Show SForm R=alice_demo
```

gives form information for the search form 'Alice\_demo' for demonstration database **Alice**.

## Show SORT

### Description

**SORT** is used in **Show** and **Print** orders to arrange output in a predefined order. Any field type except **STring** may be included in a **SORT** specification, with multiple sort fields separated by commas. The field names and types available for any particular database can be seen in the **STatus** list for that database.

Records where one or more of the sort target fields are empty will appear after all those for which the field has content, sorted in ascending record number order.

The **DESCending** modifier is used after the name of a field in a **SORT** list to arrange output alphanumerically, in descending order according to field content.

If the records in the search result were retrieved using the **FUZZ** command or the **About** function, it is useful to view them ranked in order of relevance or relative importance. To do this, use **SORT=Frequency**. Using **SOR=FRE** on a non-**FUZZy** search result displays hit records according to the number of hit terms each contains, with the record(s) containing the greatest number of hit terms shown first.

Normally, records originating from more than one database are sorted *within* the hit record set for their database, each set being sorted individually. The sorted sets are then output in the order in which the databases were opened. Including the **MERGE** qualifier in the **SORT** order merges the hit records of all databases into a single set and sorts them together. Use **Define MERGE** to make merged sorted outputs always the default within a session; use **Define NO MERGE** to cancel merged sorting.

Sorting on records which contain part records is complex. Refer to the section 'Output and Sorting' in Chapter Eight of the *TRIPclassic User Guide*, 'Head and Part Records' for further information.

### Syntax

```
Show SORT={Frequency|field[DESCending]
           [,field[DESCending][,...]] [[NO]MERGe]
```

where *field* is the key or target field on which the records will be sorted.

**Show SORT=** can also be used with the **S=**, **R=**, **Format=**, **FOcus Highlight** and **Weight** modifiers.

The **DESCending** modifier applies to each individual field used in the **SORT** order, and can be used to reverse a sorted output order, while the **weight** modifier will remove records from the output that have a percentile rank lower than 'n', applying a bottom-end cut-off to the result set.

### Examples

The following examples were taken from the **Alice** demonstration database provided with TRIP, and are to be worked consecutively.

*Example 1:*

```
S=1      <475> BAsE Alice
S=2      <8>   Find cook
Show SORT=chaptnr
```

displays the results of the most recent search (S=2) in the default format, sorted with the record with the lowest value of **chaptnr** first and the remainder in *ascending* order by **chaptnr**. Records which have identical values for **chaptnr** appear in the order in which they were added to the database.

**Example 2:**

```
S=3      <56> Find rabbit
S=4      <35> Find and white
Show SORT=person,chaptnr S=3 Format=short
```

shows the results of the third search result using the pre-defined format named 'Short', sorted first in standard alphabetical order according to the contents of the **PHrase** field **person**, and in numerical ascending order by the content of **chaptnr**. Records with the same content in the **person** field appear sorted by the **chaptnr** value. Records containing the same value in both **person** and **chaptnr** appear in the order in which they were added to the database.

**Example 3:**

```
Show SORT=chaptnr DESCending,person,speaker DESCending
Format=chaptnr,person,speaker
```

displays the contents of the **chaptnr**, **person** and **speaker** fields for the most recent search (S=4), sorted first in descending numerical order on the contents of **chaptnr**, then in standard or ascending alphabetical order by **person** and descending alphabetical order by **speaker**.

Records with identical values for **chaptnr** appear sorted alphabetically by the content of the **person** field; records containing the same values for both **chaptnr** and **person** are sorted in reverse alphabetical order by **speaker**.

**Example 4:**

This example is taken from the combined **Alice** and **Carroll** databases.

```
S=1      <499> BAsE temp=alice,carroll
S=2      <26> Find unicorn
Show Format=person SORT=person DESCending MERGe
```

shows the contents of **person** for the most recent search, with all hit records merged into a single hit record set and sorted in descending order by the content of the **person** field.

**Example 5:**

```
S=1      <99> BAsE corr
S=2      <86> FUZZ trip AND 3rip
Show SORT=FRequency
```

opens the demonstration database **Corr**, locates eighty-six records containing the terms 'trip' or '3rip', and shows them sorted by rank.

## Show USer

### Description

This order displays read and write access information for users of the TRIP system. Users are presented in alphabetical order by TRIP user name, while the databases to which each has access are listed chronologically by access date.

### Syntax

```
Show USer [R={user|ALL}]
```

where *user* is the TRIP user name for which information is being requested.

### Examples

#### Example 1:

```
Show USer
```

gives read and write access information for all users created by a user manager. If the user is SYSTEM, the access rights of users created by SYSTEM will appear. If the user is a user manager called 'Smith', access privileges for any users created by 'Smith' will be shown.

#### Example 2:

```
Show USer R=paul
```

gives access information for the user called 'Paul'. If the user needing this information is not a user manager, he or she can only get his or her own access information. If the requesting user is 'Paul', Paul's user manager/owner or SYSTEM, the user's read and write privileges will be listed.

#### Example 3:

```
Show USer R=ALL
```

presents access privilege information for all TRIP system users, and is available only to the user named SYSTEM.



## Status

### Description

The **Status** order is used to view information about a database. A database must already be open if no database name is given in the order.

The **Status** displays can be scrolled using the commands **More** and **Back**, or the <↑ **Arrow**> and <↓ **Arrow**> keys, and sent to a local printer with the **Print Local** order. They can also be written to a text file.

**Status** provides some or all of the following information:

#### Database Information

Table 26 – Database Information shown by the **Status** order

Status Screen Item	Definition
Data base	name of the database
Owner	owner of the database
Number of records	current number of records (whether added before or after the last index was done)
Design created	date and time of database design creation
Design revised	date and time of last database design modification
Last update	date and time of last database update
Last index	date and time of last database index update
Description	description of the database
Default format	the default output format specified for this database
Default entry form	the default data entry form specified for this database
Formats available	the output formats available for this database
Entry forms available	the data entry forms available for this database
Files	database file locations
Notify on completion	notify user of batch job completion

## Field Information

Table 27 – Field information shown by the **Status** order

Column Item	Definition
Field name	name of the field
No	number of the field
Type	type of field, i.e. TExt, PHrase, DAte, TIme, INteger, NUmber or STring
Part	whether or not the field is a part field (Y/N)
Mand	whether or not the field is mandatory (Y/N; if Y, it cannot be empty)
Indx	whether the field is indexed (Y), not indexed (N), word indexed (W) or separately indexed (S)
Orig	whether or not the original layout of the input to the field is retained (Y/N)
Cost	the unit cost for showing and printing the contents of a field
Comment	a short description or comment about the field

## Field List

Table 28 – Field list information shown by the **Status** order

Row Item	Definition
Field name	name of the field
Record name	a field defined as the record name field
Record number	a field defined as the record number field
Part record name	a field defined as the part record name field
Copyright protected field	a field whose content is copyright protected
Copyright holder field	a field containing the copyright holder's name (usually the author) of the contents of the copyright protected field
ASE before storing TForm records	the name of an ASE to be called before storing TForm records
ASE after storing TForm records	the name of an ASE to be called after storing TForm records

## Syntax

`Status [database]`

where *database* is the name of the database whose status is being requested.

## Examples

*Example 1:*

`Status`

displays status information for the database that is currently open.

*Example 2:*

`Status Alice`

gives status information for the TRIP demonstration database **Alice**.

*Example 3:*

```
Print BAsE R=Alice FILE=alice.sta
```

writes the **Status** of database **Alice** to a text file called 'Alice.sta'.

*Example 4:*

```
Status carroll
```

```
Print Local
```

brings the first status screen for demonstration database **Corr** to the screen, then sends the entire status text to a local printer.



## STOP

### Description

**STOP** ends a TRIP session. If **STOP** is used alone, the session search history will not be saved. If the modifier **SAve** is included, everything that affects the current search session (the *search environment*) is retained, so that the next session is started with the search history exactly as it was when the **STOP** order was given.

### Syntax

```
STOP [[SAve] [[FILE=file] [[NO] HIghlight]]]
```

#### Note:

*The modifiers **FILE=** and **[NO] Highlight** are intended for creating saved Session Information Files (SIF) to be used with the **Find SScope** and **UPDate SScope** commands.*

### Examples

#### Example 1:

```
STOP
```

ends the TRIP session, discarding *all* search results of the current session.

#### Example 2:

```
STOP SAve
```

ends the TRIP session with saving. When the next TRIP CCL search session is started, *all* orders saved as part of the previous search session will be shown automatically in the CCL window, exactly as they appeared when the order **STOP SAve** was given.

#### Example 3:

```
STOP Save FILE=bigdata.sif
```

ends the TRIP session, saving the current session information in a file named "*bigdata.sif*".

#### Example 4:

```
STOP Save NO HIghlight
```

ends the TRIP session, saving the current session information without hit highlighting.

#### Example 5:

```
STOP Save FILE=nohighlights.sif NO HIghlight
```

ends the TRIP session, saving the current session information without hit highlighting, in a file named "*nohighlights.sif*".

## Top

### Description

**Top** moves the cursor to the top of the current record during a **Show** order. If the search result contains more than one record, the cursor moves to the top line of the record currently on display.

### Syntax

Top



## TRace

### Description

The **TRace** order is used to view the history of one particular search or selection of searches, showing all previous searches used to form the particular search specified.

If no search number or range of search numbers is given, the default is S=0, or the most recent search.

### Syntax

```
TRace [S=range]
```

where *range* is a search result number or series of searches to be traced.

### Examples

Consider these orders as having been given consecutively.

#### Example 1:

```
S=1      <24>  BAsE carroll  
S=2      <16>  Find king OR queen  
S=3      <10>  Find R=TO 10  
S=4      <13>  Find hare OR red  
S=5      <17>  Find OR S=2
```

```
TRace
```

- opens the demonstration database **Carroll** (S=1)
- finds the records containing the terms 'king' or 'queen' (S=2)
- finds the first ten records in the database (S=3)
- finds the records containing the terms 'hare' or 'red' (S=4)
- finds all the records of search results two and four (S=5)
- and traces the essential search orders leading to (and including) the most recent search (S=0), in this case search numbers five, four, two and one.

#### Example 2:

```
TRace S=3
```

traces the essential search orders leading to (and including) search number three, which are search results three and one.

#### Example 3:

```
TRace S=4 TO 5
```

traces the essential search orders leading to (and including) search result numbers four and five.

## UPDate

### Description

This order is used for global updating to replace a given word, sentence, paragraph, subfield or field in several records within a database simultaneously.

It may also revise the contents of the **SDI** (Selective Dissemination of Information) timestamp and record number fields.

### UPDate – Section Index

**UPDate** is covered under the following headings:

Table 29 – List of UPDate orders

Command	Subject
UPDate	Global Updating
UPDate	SDI

### Syntax

To update records in a search set:

```
UPDate [COpy[=database]]{SUBField|PARagraph
|SENtence|WORD|field[.{subfieldno
|paragraphno[.sentenceno]}}]=newdata WHere S=n
```

To update records specified by number:

```
UPDate [COpy[=database]]field[.{subfieldno
|paragraphno[.sentenceno]}]=newdata
WHere R=range
```

To copy modified records:

```
UPDate COpy[=database] WHere {S=n|R=range}
```

To update SDI markers:

```
UPDate SDI [database[{R=n|TStamp=date[time]}]]
```

where *database* is the name of the database to receive new data, *newdata* the new material to be inserted, *n* a record or search result number and *range* a single record number or series or range of record numbers.

The modifiers are restricted by field type, as shown below:

Table 30 – List of UPDate order modifiers

Modifier	Applicable Field Type
SUBField	PHrase only
PARagraph	TExt only
SENtence	TExt only
WORD	TExt and PHrase

## UPDate – Global Updating

### Description

This order is used for global updating to replace a given word, sentence, paragraph, subfield or field in several records within a database simultaneously.

The qualifier **Where** followed by **S=** and a search result number states which search results should be modified. When followed by **R=** and a record number or list or range of record numbers, **Where** indicates which records should be updated. Ranges of record numbers can be specified using **TO**, and **FRom**. Ranges and values are separated by commas.

**UPDate COPY** allows copies of selected records to be optionally modified, then added to the current or other database.

### Syntax

To update records in a search set:

```
UPDate [COPY[=database]]{SUBField|PARagraph
|SENTence|WORD|field[.{subfieldno
|paragraphno[.sentenceno]}}]=newdata Where S=n
```

To update records specified by number:

```
UPDate [COPY[=database]]field[.{subfieldno
|paragraphno[.sentenceno]}}]=newdata
Where R=range
```

To copy modified records:

```
UPDate COPY[=database] Where {S=n|R=range}
```

where *database* is the name of the database to receive new data, *newdata* the new material to be inserted, *n* a search result number and *range* a single record number or series or range of record numbers.

**Note:**

*Global updating with the WORD modifier will replace only the last word in a replacement string. This is because when global updating, only one word - the last word - will be identified for updating from a complex search term with Boolean operators belonging to the AND class.*

*E.g. A search for Meredith AND Carl OR John Haynes, followed by UPDate WORD=John Henry Where S=0, will lead to occurrences of John John Henry where John Haynes was found earlier, and of John Henry wherever the word Carl occurred in the same record as the word Meredith.*

*It is therefore recommended that the WORD modifier be avoided when global updating search sets emanating from complex searches.*

### Examples

The following examples are intended for use with the demonstration database **Corr**.

**Example 1:**

```
S=1      <99> BAsE corr
UPDate rname=matthew Where R=FRom 75
```



opens the demonstration database **Corr**, deletes the entire contents of **rname** in all records numbered seventy-five or higher (even if there is more than one subfield) and replaces them with a single subfield containing the term 'Matthew'.

**Example 2:**

```
UPDate rcomp.1="Matthew Industries, Inc."      WHere R=70 TO
74
```

replaces the contents of the first subfield of the **rname** field with 'Matthew Industries, Inc.', in **Corr**'s records seventy through seventy-four inclusive.

**Example 3:**

```
S=2      <56> Find PHrase=mats
UPDate SUBField="Matthew Longley" WHere S=0
```

finds all records containing the term 'mats' in any **PHrase** field, and replaces the hit subfield with 'Matthew Longley'.

**Example 4:**

```
UPDate content.1.1="New Data:" WHere R=4
```

substitutes the first sentence of the first paragraph of the **Text** field **content** with 'New Data:' in record number four.

**Example 5:**

```
S=3      <5> Find telex
UPDate SENTence="Thank you for your FAX."      WHere S=0
```

replaces each **Text** field sentence located by the most recent search result number with 'Thank you for your FAX.'

**Example 6:**

**Note:**

*The result will not be as expected*

```
S=4      <2> Find TExt=john
S=5      <1> Find john haynes
UPDate WORD="John Henry" WHere S=5
```

*replaces every instance of 'Haynes' found by search number five with 'John Henry', hence 'John Haynes' would now read, 'John John Henry'.*

**Example 7:**

```
UPDate rname="John Smith" WHere R=4,5,7
```

substitutes each subfield of every **PHrase** field of record numbers four, five and seven with the phrase 'John Smith'.

**Example 8:**

```
S=6      <5> Find day=198502
UPDate SUBField=19940201 WHere S=0
```

overwrites each subfield where the **Date** field **day** contains 'February 1985' with the date 'February 1, 1994'.

**Example 9:**

```
S=7      <1>   Find rname=mats and rolf
UPDate rname.2="John Smith" WHere S=0
```

changes the value in the second subfield of the **PHrase** field **rname** to 'John Smith', in every record located by the most recent search result.

**Example 10:**

```
S=7      <1>   Find germany
UPDate COPy=corr Word=sweden WHere S=2
```

exchanges the word 'Germany' found by search number two with the term 'Sweden' and adds the modified record to **Corr**, the database defined by the **UPDate COPy** order as the copy destination.

**Example 11:**

```
S=8      <60>  Find mats
UPDate COPy=corr_save WHere S=0
```

copies records containing the term 'mats' from **Corr** to **Corr\_save** without modification. **Corr\_save** must already exist

## UPDate SCoPe

### Notes:

*The “UPDate SCoPe” modifier is NOT a part of Global updating.*

*“UPDate SCoPe” is related to the “Find SCoPe” modifier.*

*Neither “UPDate SCoPe” nor “Find SCoPe” are connected with, or influenced by, the “Define SCoPe” modifier.*

### Description

This function is used to update a previously saved, predefined search set that is to be used for searching across large database clusters of mostly static data.

### Syntax

```
UPDate SCoPe(databaseName)
```

where *databaseName* is the name of the database in which the search sets are to be updated.

#### Example:

```
UPDate scope(db12)
```

this updates the search sets with the result of new searches for db12; no new search sets are created, only the old ones are updated.

#### Note:

*For more information, see **Appendix B - Scope Search Facility of the TRIPmanager Administration guide**.*

## UPDate SDI

### Notes:

*The “UPDate SDI” modifier is NOT a part of Global updating.*

*“UPDate SDI” is related to the “Define SCoPe” modifier.*

### Description

TRIP’s **SDI** (**S**elective **D**issemination of **I**nformation) feature allows users to selectively review records, both by record number and date and time of last index. The **UPDate SDI** order acts as command and modifier in revising the contents of the **SDI** (**S**elective **D**issemination of **I**nformation) timestamp and record number fields.

Refer to the ‘**DEfine SCoPe**’ section in this guide for further information regarding **SDI**.

### Syntax

```
UPDate SDI [database{R=n|TStamp=date[time]}]]
```

where *database* is the name of the database to revise, *n* is a record number with which to update the SDI record marker, and *date* and *time* are date and time values respectively.

### Examples

The following examples were written for use with the demonstration database **Corr**.

#### Example 1:

```
S=1      <99>  BAsE corr
S=2      <99>  DEfine SCoPe SDI
UPDate SDI
```

opens database **Corr**, begins an **SDI** session, limits the scope to records that have been added or modified (and subsequently indexed) since the last **UPDate SDI**, and updates the user record to indicate that the records just searched will not be included in the next **SDI** run. This order updates the **SDI** timestamp marker with the date and time that **Corr** was last indexed.

#### Example 2:

```
UPDate SDI corr R=48
```

updates the current highest record number in **Corr** to number forty-eight. TRIP now includes records numbered forty-nine and above in the scope defined by the next **DEfine SCoPe SDI NO UPDate** order.

#### Example 3:

```
UPDate SDI corr TSTAMP=19940328
```

causes TRIP to regard all records in database **Corr** that have been indexed after the twenty-eighth of March, 1994 as new in the next **SDI** run.

## List of Figures and Tables

### List of figures

Figure 1 – The TRIPmanager CCL Query Window .....	11
Figure 2 – The TRIPclassic CCL Command window .....	12
Figure 3 – TRIPclassic Numeric Keypad .....	13
Figure 4 – TRIP Manager Display term list .....	94
Figure 5 – TRIP Manager Display list with items selected .....	95

### List of Tables

Table 1 – DEfine orders.....	21
Table 2 – Operators for the order DEfine AND .....	27
Table 3 – Modifiers for the order DEfine Hlghlight .....	44
Table 4 – Default truncators and masks for the FIND order .....	52
Table 5 – Default maxima and minima for the DEFINE order .....	55
Table 6 – Elements of the order DEfine THESaurus.....	75
Table 7 – Effect on CCL searching of the DEfine TIMEform order .....	77
Table 8 – List of DElete order sections .....	85
Table 9 – List of Display order sections .....	92
Table 10 – List of EDit orders .....	114
Table 11 – List of FIND order variants .....	126
Table 12 – List of FIND AND operators .....	129
Table 13 – List of logical NOT conditions.....	133
Table 14 – Default system search masking symbols .....	138
Table 15 – User of the \$ symbol in search masking .....	138
Table 16 – Proximity search operator usage .....	140
Table 17 – Relational operators for the Find order.....	151
Table 18 - Use of Short form operators with the Fuzz command .....	177
Table 19 – Commands for which online help is available.....	180
Table 20 – Operators for which online help is available .....	180
Table 21 – Symbols for which online help is available .....	180
Table 22 – Modifiers & qualifiers for which online help is available .....	181
Table 23 – INSert order modifier restrictions by field type.....	186
Table 24 – List of Print orders.....	196
Table 25 – List of SHow orders .....	240
Table 26 – Database Information shown by the SStatus order .....	265
Table 27 – Field information shown by the SStatus order .....	266
Table 28 – Field list information shown by the SStatus order.....	266
Table 29 – List of UPDate orders.....	271
Table 30 – List of UPDate order modifiers .....	271

## Main Index

### Start of Main Index

! symbol.....	53, 137	TRIP Manager .....	11
# symbol.....	53, 137	<i>TRIPclassic</i> .....	13
\$ symbol.....	137, 139	order	
\$\$ symbol.....	137	maximum length .....	10
\$\$\$ symbol.....	137	search	
& symbol .....	53	menu option.....	8
* symbol .....	53	CENTury MINimum	
. symbol.....	139	DEfine.....	29
... symbol.....	139	Chevron	
/ symbol.....	53	as convention.....	8
: symbol.....	53	Close .....	18
; symbol.....	53	Comma	
? symbol.....	53	DEfine.....	24
@ symbol.....	53	and AND .....	24
_ symbol.....	53	AND.P .....	25
` symbol.....	53	AND.S .....	24
<Enter> .....	8	and Boolean operators .....	24
<Gold>		and comma character .....	25
as convention .....	8	and OR .....	24
<Leave>		and proximity operator .....	24
as convention .....	8	Comma character	
<Next page>.....	8	and SAVE.....	234
<Next>		COMmand KEY	
as convention .....	8	DEfine.....	30
<Page down> .....	8	Continue .....	19
<Page up> .....	8	DEfine.....	31
<PF1> .....	8	CONTinue	
<PF3> .....	8	and Show.....	247
<Prev>		Show.....	247
as convention .....	8	Conventions	
<Previous page> .....	8	<Gold> .....	8
<Return> .....	8	<Leave>.....	8
ABout		<Next> .....	8
DEfine .....	26	<Prev> .....	8
ACcess		<i>boldface</i> .....	8
Print .....	199	chevrons .....	8
AND .....	128	Courier fonts .....	8
DEfine .....	27	italic .....	8
Back .....	14	lower case.....	8
command		space character .....	8
and Show Focus .....	192, 193	upper case .....	8
BASe.....	15	COPy	
Print .....	200	DEfine.....	32
Show.....	243	COST	
<b>Bold</b>		Show.....	245
as convention .....	8	Courier fonts	
BT .....	110, 163	as convention.....	8
CALi .....	17	CT .....	110, 163
CCL		Display.....	111
and		CUt	

## Main Index

DEfine .....	33	MAP .....	50, 165
Data type .....		and search terms .....	50
INteger .....	150, 191	and source database .....	50
NUmber .....	191, 209	and source field .....	50
TIme .....	209	and source records .....	50
Database .....		and target database .....	50
<b>information</b> .....		and Vlew names/types .....	50
<b>and SStatus</b> .....	264	and virtual fields .....	50
simultaneously open .....		MASK .....	52
maximum number .....	15	MAXimum/MINimum .....	55
source .....	165	MERGE .....	57
Default .....		NO .....	
Display .....	98	FOcus .....	39
sort order for Display list .....	108	HOLD .....	46, 228
TIMEForm .....	77	MERGE .....	57
DEfine .....	21, 23	REVerse .....	62
ABout .....	26	NO Hlghlight .....	44
AND .....	27	PAge .....	58
and start-up procedures .....	21	PCode .....	59
CENTury MINimum .....	29	Print .....	
comma .....	24	MAXimum .....	56
and AND .....	24	PRINTER .....	61
AND.P .....	25	REVerse .....	62
AND.S .....	24	Save BASe .....	64
and Boolean operators .....	24	SCope .....	65
and comma character .....	25	and Display .....	65
and OR .....	24	and Find .....	65
and proximity operator .....	24	SCope SDI .....	66
COMmand KEY .....	30	removing, with DEfine SCope .....	66
command word .....	8	SORT .....	
Continue .....	31	MAXimum .....	56
COpy .....	32	SPace .....	69
CUt .....	33	= 72 .....	
Display .....		and . . . .....	70
MAXimum .....	56	and AND .....	69
EForm .....	36	AND .....	72
Find .....		AND.P .....	70
MINimum .....	56	AND.S .....	69
FOcus .....	39	and proximity operator .....	69
and Expand .....	39, 121	STop WORD .....	74, 176
and Show .....	247	THESaurus .....	75, 79
Format .....	40	TIMEForm .....	77
and named output formats .....	40	Vlew .....	80
and predefined output formats .....	40	WEIght .....	82
and Print .....	207	DEfine? .....	55
and run-time output formats .....	40	DElete .....	84
FUZZ .....	37, 42, 78	and index update .....	84
Hlghlight .....	44	and most recent search .....	84
default .....	44, 252	and procedures .....	84
HOLD .....	46	and saved searches .....	84
KEY .....	47, 48	global .....	84
LPcode .....	49	operations on BASe file .....	84

## Main Index

Print .....	227	type	
WHere R=n .....	89, 185, 271	INteger .....	150, 191
WHere S=n .....	89, 185, 271	NUmber .....	191, 209
Direct searching		TIme .....	209
on numerical contents		virtual .....	165
relational operators .....	150	and DEfine MAP .....	50
on record number .....	154, 156	FILE	
on textual contents		Print .....	204
field qualifiers .....	134	Find .....	124, 126
proximity searching .....	139	ABout .....	170
Display .....	91, 97	and	
and DEfine SCoPe .....	65	Help .....	180
CT .....	111	and DEfine	
default .....	98	SCoPe .....	65
field qualifiers .....	98	Class .....	171
FUZz .....	177, 178	Duplicate .....	160
FUZz() .....	104	Execution order .....	146
list .....	84, 93, 95	FUZz .....	148
NT .....	111	Previous Searches .....	142
previous search results .....	100, 106, 116	SAVE .....	156
Print .....	202	Terms From a Display List .....	144
RT .....	111	Thesaurus .....	163
sort on frequency .....	107, 108	TStamp .....	157, 158
SORT=FRequency .....	107, 108	USer .....	161, 162
thesaurus modifiers .....	110	FOCUS	
UP .....	111, 112	and Show .....	247
Edit .....	113	as qualifier for Show order .....	39
INSert .....	117	DEfine .....	39
EDit		and Expand .....	39, 121
active record .....	115	and Show .....	247
EDit COpy .....	119	Format	
EDit INSert .....	117	and Print .....	207
EDit PART .....	118	and Show .....	248
EDit SORT .....	120	DEfine .....	40
EForm		FRequency	
and EXPort .....	123, 183	and Display SORT= .....	107, 108
DEfine .....	36	FRequency .....	173
Print .....	203	Print .....	209
Show .....	246	FRom	
Expand .....	121	and SAVE .....	234
and DEfine FOCus .....	39, 121	FUZz .....	176
and FOCus .....	121	DEfine .....	37, 42, 78
EXPort .....	122	Display .....	177, 178
EForm .....	123, 183	Global deletion orders .....	84, 86, 87, 88, 89
PRocedure .....	123, 183	GRoup	
arguments concatenation .....	122	Print .....	210
External transaction sets .....	168	Show .....	251
Field		Help .....	179
<b>information</b>		Find .....	180
<b>and SStatus</b> .....	265	HIDe .....	181
qualifiers .....	134	Highlight	
source .....	165	Show .....	252



## Main Index

Highlight		\$\$\$ .....	137
DEfine .....	44	& 53	
Print .....	211	* 53	
HOLD		/ 53	
and Print .....	227	: 53	
DEfine .....	46	; 53	
IMPOrt .....	182	? 53	
INDex .....	184	@ .....	53
Indirect searching		53	
on textual contents		53	
transaction sets		MAXimum	
external .....	168	DEfine .....	55
internal .....	167	Display .....	56
mapped .....	165	Print .....	56
INSert .....	185	SORT .....	56
INteger .....	150, 191	MEasure .....	191
Internal transaction sets .....	167	Print .....	215
Italic		MERGE	
as convention .....	8	as qualifier for SORT .....	57
KEY		DEfine .....	57
DEfine .....	47, 48	MINimum	
LEAve .....	188	DEfine .....	55
List		Find .....	56
Display .....	84, 93, 95	More .....	192
LlSt .....	189	Next .....	193
Print .....	212	NO	
LOAD .....	190	FOcus	
Local		DEfine .....	39
Print .....	213	HOLD	
Logical		DEfine .....	46
operator		MERGE	
AND .....	128	DEfine .....	57
NOT .....	132	REVerse	
OR .....	130	DEfine .....	62
XOR .....	131	NOT .....	132
Lower case		NOW	
as convention .....	8	and Print .....	227
LPcode		NR .....	110
DEfine .....	49	NT .....	110, 163
MAP		Display .....	111
DEfine .....	50, 165	Number	
Mapped transaction sets .....	165	record	
MASK		SDI marker	
DEfine .....	52	and newly-added records .....	66
Masking		NUMber .....	191, 209
and direct searching on textual contents: 135,		Operator	
136		logical	
symbol		AND .....	128
! 53, 137		NOT .....	132
# 53, 137		OR .....	130
\$ 137		XOR .....	131
\$\$\$ .....	137	proximity	

## Main Index

\$	139	TForm	225
.	139	TRace	229
...	139	USer	230
relational	150	WAIT	227, 228
Operators		Print?	228
Boolean		PRINTER	
and DEfine comma	24	DEfine	61
relational		Print	217
searching on numerical contents	150	Procedure	
OR	130	Print	218
PAGE		Show	254
DEfine	58	Procedures	
PART SORT		and EXPORT	123, 183
Print	216	and Run	233
Show	253	from SAVE orders	234
PCode		Proximity operator	
DEfine	59	\$	139
PREVious	194	.	139
Print	195, 198	...	139
ACcess	199	Proximity searching	139
and DEfine		Record	
Format	207	number	154, 156
NO		and Show	256
HOLD	228	SDI marker	
BASE	200	and newly-added records	66
DELeTe	227	Relational operator	
Display	202	searching on numerical contents	150
EForm	203	Relational operator	150
FILE	204	RENumber	231
UNIX host	205	REVeal	232
Windows host	205	REVerse	
FOcus	206	and Show	62, 220, 258
Format	207	DEfine	62
FREquency	209	RT	110
GRoup	210	Display	111
HIghlight	211	Run	233
HOLD	227	and procedures	233
in reverse order	220	and saved searches	233
LlSt	212	SAVE	
Local	213	and comma character	234
LOCAL	195	and DELeTe	234
MEasure	215	and FROM	234
NOW	227, 228	and TO	234
PART SORT	216	SAVe	234
previous searches	221	SAVe BASE	
PRINTER	217	DEfine	64
PRocedure	218	SCOpe	
record numbers	219	DEfine	65
SForm	222	Scope note	110
SORT	223	SCOpe SDI	
and SStatus	223, 261	DEfine	66
specific formats	207	SDI	

## Main Index

marker		
record number		
and newly-added records .....	66	
Timestamp .....	66	
and newly-added and modified records .....	66	
UPDate .....	66	
SDI - Selective Dissemination of Information	66	
Search		
execution order .....	146	
orders, saved as procedures .....	234	
previous, and Show .....	259	
saved		
and DELeTe .....	234	
saved, and Run .....	233	
Searching		
direct		
on numerical contents		
relational operators .....	150	
on record number .....	154, 156	
on textual contents		
field qualifiers .....	134	
proximity searching .....	139	
indirect		
on textual contents		
transaction sets		
external .....	168	
internal .....	167	
mapped .....	165	
proximity .....	139	
Searching on textual contents .....	163	
SElect .....	236	
SForm .....	238	
Print .....	222	
Show .....	260	
Show .....	239, 241	
and CONTInue .....	247	
and DEfine		
Focus .....	247	
and FOCUS qualifier .....	39	
and run-time SORT .....	261	
and system information .....	239	
BASE .....	243	
COST .....	245	
EForm .....	246	
FOCUS .....	247	
and BACK .....	192, 193	
<i>Format</i> .....	249	
GRoup .....	251	
Hlghlight .....	252	
PART SORT .....	253	
previous searches .....	259	
PRocedure .....	254	
record numbers .....	256	
REVerse .....	258	
SForm .....	260	
SORT .....	261	
and MERGe .....	57	
specific formats .....	248	
USer .....	263	
Sign: .....	150	
SN .....	110	
SORT		
=FRequency .....	107, 108	
and Print .....	223	
and Show .....	261	
Source database .....	165	
Source field .....	165	
SPace		
DEfine .....	69	
and		
. . 70		
and AND .....	69	
AND .....	72	
AND.P .....	70	
AND.S .....	69	
and proximity operator .....	69	
SPace =		
DEfine .....	72	
Space character		
as convention .....	8	
Status .....	264	
and Print SORT .....	223, 261	
and Show .....	248	
database		
entry forms available .....	264	
output formats available .....	264	
field		
comment .....	265	
index .....	265	
mandatory .....	265	
original .....	265	
part .....	265	
record		
name .....	265	
number .....	265	
<b>information</b>		
<b>database</b> .....	264	
<b>field</b> .....	265	
STOP .....	267	
and Print .....	228	
Stop Word		
DEfine .....	74	
Symbol		

## Main Index

masking		DEfine .....	75, 79
! 53, 137		<b>narrower term:</b> .....	110
# 53, 137		NR .....	110
\$ 137		NT .....	110
\$\$ .....	137	<b>related term:</b> .....	110
\$\$\$ .....	137	RT .....	110
& 53		SN .....	110
* 53		UF .....	75, 110
/ 53		Time .....	209
: 53		TIMEForm	
; 53		default .....	77
? 53		DEfine .....	77
@ .....	53	TIMEForm=1 .....	77
53		TIMEForm=3 .....	77
53		Timestamp	
Symptom - field in database Cause&Effect ..	51	SDI marker .....	66
Tab		and newly-added and modified records ..	66
as convention .....	8	UPDate SDI .....	67
Term		TO	
'child' .....	110	and SAVE .....	234
'parent' .....	110	Top .....	268
'sibling' .....	110	TRace .....	269, 271, 275
broader .....	110	Print .....	229
controlled .....	110	Transaction sets	
narrower .....	110	external	
related .....	110	indirect searching on textual contents: .	168
top		internal	
in thesaurus .....	110	indirect searching on textual contents: .	167
used for .....	110	UF .....	75, 110
Term number .....	110	UP .....	163
TForm		Display .....	111, 112
Print .....	225	UPDate .....	270
Thesaurus		SDI .....	66
'used for' term .....	110	Timestamp .....	67
broader term .....	110	Upper case	
controlled term .....	110	as convention .....	8
definition of .....	110	User	
narrower term .....	110	Print .....	230
netlike structure .....	110	Show .....	263
related term .....	110	View	
scope note .....	110	DEfine .....	80
structure .....	110	Virtual field .....	165
term number .....	110	and DEfine MAP .....	50
top term .....	110	WAIT	
treelike structure .....	110	and Print .....	227
THESaurus		WEight	
<b>broader term:</b> .....	110	DEfine .....	82
<b>controlled term:</b> .....	110	When to Print .....	227
CT .....	110	XOR .....	131