



digital  
vision  
group

# **TRIP Administration with TRIPmanager**

TRIPsystem  
Product Documentation

## End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether “Software”) supplied by 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 unfringed and protected.



## Table of Contents

<b>ABOUT THIS GUIDE .....</b>	<b>9</b>
SCOPE AND ASSUMPTIONS .....	9
THE TRIP DOCUMENTATION LIBRARY .....	9
THE STRUCTURE OF THIS GUIDE .....	10
CONVENTIONS USED IN THIS GUIDE .....	11
TRIP NAMING CONVENTIONS .....	12
TRIP LOGICAL NAMES .....	12
<b>PART 1: DATABASE ADMINISTRATION .....</b>	<b>13</b>
<b>CHAPTER 1: FUNDAMENTALS .....</b>	<b>14</b>
NAVIGATION WITHIN TRIPMANAGER .....	14
TRIP SYSTEM BASICS .....	15
<i>Introduction</i> .....	15
<i>Data Models</i> .....	15
<i>Data Organisation</i> .....	17
<i>TRIP Field Types</i> .....	18
<i>The CONTROL Database</i> .....	20
<i>TRIP Manager Privileges</i> .....	20
TRIP DATABASE BASICS .....	20
<i>Records</i> .....	20
<i>File Structures</i> .....	23
<b>CHAPTER 2: DATABASES .....</b>	<b>25</b>
NOTES ON FILE LOCATIONS .....	25
CREATING THE DATABASE .....	25
GENERAL DATABASE PROPERTIES .....	26
<i>Database Name</i> .....	26
<i>Physical File Locations</i> .....	27
<i>Transaction Log</i> .....	27
<i>XML Enabling the Database</i> .....	29
<i>Description of the Database</i> .....	29
<i>Saving the database design</i> .....	30
MODIFYING DATABASE PROPERTIES .....	31
<i>Database Properties (1) – General</i> .....	31
<i>Database Properties (2) – Files</i> .....	34
<i>Database Properties (3) – Indexing</i> .....	36
<i>Database Properties (4) – Links</i> .....	50
<i>Database Properties (5) – Advanced</i> .....	52
FIELD DEFINITION .....	55
<i>Defaults and Restrictions</i> .....	55
<i>The Modify Fields Collection Form</i> .....	55
SAVING A FIELD DESIGN .....	70
COMMITTING FIELD DESIGNS AND CHANGES TO THE DATABASE .....	70
DELETING A FIELD DESIGN .....	71
SAVING A DATABASE DESIGN .....	71
MODIFYING A DATABASE DESIGN .....	71
DELETING A DATABASE DESIGN .....	72
COPYING A DATABASE DESIGN .....	72
RELATED CCL COMMANDS .....	73
<i>Status</i> .....	73
<i>Show</i> .....	74

## Table of Contents

Print .....	74
IMPORT and EXPORT .....	74
DATABASE CLUSTERS .....	75
Creating a Cluster .....	75
Modifying a Database Cluster .....	76
Deleting a Cluster .....	78
RELATED CCL COMMANDS .....	78
SINGLE NAMESPACE .....	79
<b>CHAPTER 3:     THESAURI .....</b>	<b>80</b>
WHAT IS A THESAURUS? .....	80
A SIMPLE THESAURUS .....	81
CREATING A THESAURUS .....	83
THESAURUS STRUCTURE .....	84
Data Layout .....	84
THESAURUS DATABASE DESIGN .....	87
General Thesaurus Properties .....	87
Special Thesaurus Fields .....	87
Defaults .....	87
Character Sets .....	87
Description of the Thesaurus .....	87
Other Thesaurus Properties .....	87
Field Definition .....	87
FILLING THE THESAURUS .....	88
Using TForm .....	88
Using Data Entry .....	88
RELATED CCL COMMANDS .....	88
Status .....	88
Show .....	89
IMPORT/EXPORT .....	89
<b>CHAPTER 4:     SYSTEM LOGGING FUNCTIONS .....</b>	<b>90</b>
OVERVIEW .....	90
ACTIVATING SYSTEM ACCOUNTING FUNCTIONS .....	90
Assigning Field Costs for Accounting .....	90
Accounting function Logical Names .....	90
Accounting Log File Format .....	92
EVENT LOGGING .....	95
Overview .....	95
How to Enable Event Logging .....	96
Parameters .....	96
Event Log Output .....	97
Log File Location and Name .....	98
<b>PART 2: FORMS .....</b>	<b>100</b>
<b>CHAPTER 5:     TRIPCLASSIC DATA ENTRY FORMS .....</b>	<b>101</b>
CREATING AND MODIFYING TRIPCLASSIC DATA ENTRY FORMS .....	102
COPYING TRIPCLASSIC DATA ENTRY FORMS .....	102
DELETING TRIPCLASSIC DATA ENTRY FORMS .....	103
<b>CHAPTER 6:     REPORTS / OUTPUT FORMATS .....</b>	<b>105</b>
THE REPORT .....	105

## Table of Contents

COPYING REPORTS .....	107
DELETING REPORTS .....	107
CREATING A NEW REPORT .....	107
<i>Defining Layout Boxes</i> .....	110
<i>Background Text</i> .....	120
<i>Functions</i> .....	126
<i>Page Control</i> .....	140
OUTPUT FORMATS FOR DATABASE CLUSTERS .....	142
RELATED CCL COMMANDS .....	143
OUTPUT FORMAT REFERENCE GUIDE .....	144
<APPEND> .....	144
<AT_BEGIN> .....	146
<AT_END> .....	147
<BASE> .....	148
<CALL>—Format .....	149
<CALL>—Text String .....	151
<CASE> .....	152
<CHARSET> .....	154
<CHR> .....	155
<CLASS> .....	156
<CURDATE> .....	157
<DATEFORM> .....	158
<DEBIT> .....	160
<ENTITIFY> .....	161
<FF> .....	162
<FOR> Loops .....	164
<HITLIST> .....	167
<HITS> .....	169
<IF-CHANGED> .....	170
<IF-EMPTY> .....	172
<IF-NONEMPTY> .....	173
<IF-UNCHANGED> .....	174
<INDENT> .....	176
<LINK> .....	178
<NOFF> .....	181
<NOLF> .....	182
<NOORIG> .....	183
<NUMFORM> .....	185
<OCCS> .....	187
<ONCE> .....	188
<ORIG> .....	189
<PAGENO> .....	191
<PARTS> .....	192
<RID> .....	193
<RIS> .....	194
<RNAME> .....	195
<SORTFIELDS> .....	196
<SUBRID> .....	197
<SUBSTRING> .....	198
<Text Variables> .....	199
<TRACE> .....	200
<TIMEFORM> .....	201
<WEIGHT> .....	202

## Table of Contents

<b>CHAPTER 7: SEARCH FORMS.....</b>	<b>203</b>
CREATING AND MODIFYING TRIPCLASSIC SEARCH FORMS .....	204
COPYING TRIPCLASSIC SEARCH FORMS .....	204
DELETING TRIPCLASSIC SEARCH FORMS .....	205
<b>PART 3: BATCH UPDATE .....</b>	<b>206</b>
<b>CHAPTER 8: GLOBAL UPDATING .....</b>	<b>207</b>
COMMAND OVERVIEW.....	207
<i>Updating Using Record Numbers.....</i>	<i>208</i>
<i>Updating Using a Search Result.....</i>	<i>211</i>
GLOBAL UPDATING OF PART RECORDS .....	215
COPYING WITH GLOBAL UPDATE .....	215
CASE SENSITIVITY .....	216
THE LOG FILE .....	217
ERROR CHECKING.....	217
<b>CHAPTER 9: LOADING, INDEXING AND REINDEXING.....</b>	<b>218</b>
INDEX .....	218
LOAD AND LOAD/INDEX .....	219
CHECKING THE RESULTS.....	220
ERROR LOGGING .....	220
REINDEXING A DATABASE .....	220
WHEN BATCH JOBS FAIL.....	221
<i>On UNIX and Windows systems .....</i>	<i>221</i>
<i>On UNIX systems only .....</i>	<i>221</i>
<b>PART 4: DATABASE SECURITY .....</b>	<b>222</b>
<b>CHAPTER 10: USER PRIVILEGES.....</b>	<b>223</b>
TRIP'S INTERNAL ACCESS PRIVILEGES.....	223
<i>The TRIP System Manager .....</i>	<i>223</i>
<i>The TRIP 'Superman' Logical Name .....</i>	<i>223</i>
<i>TRIP File and User Managers .....</i>	<i>223</i>
<i>The TRIP User Group .....</i>	<i>223</i>
<i>The Individual or End User in TRIP .....</i>	<i>224</i>
CREATING A NEW TRIP USER.....	225
<i>Deleting a TRIP User .....</i>	<i>226</i>
<i>User Properties .....</i>	<i>227</i>
<i>User Properties (1) – General.....</i>	<i>228</i>
<i>User Properties (2) – Procedures.....</i>	<i>230</i>
<i>User Properties (3) – Groups.....</i>	<i>231</i>
<i>User Properties (3) – Access Rights.....</i>	<i>232</i>
CREATING A USER GROUP.....	232
<i>Deleting a User Group.....</i>	<i>234</i>
<i>Adding a Group Member .....</i>	<i>235</i>
<i>Deleting a Group Member .....</i>	<i>235</i>
TRANSFERRING USER RESPONSIBILITY.....	236
RELATED CCL COMMANDS .....	237
<i>Show .....</i>	<i>237</i>
<i>Print .....</i>	<i>237</i>
<b>CHAPTER 11: ACCESS RIGHTS .....</b>	<b>238</b>

## Table of Contents

DATABASE ACCESS RIGHTS DEFINITION .....	239
<i>Database</i> .....	239
<i>User / Group</i> .....	239
<i>General Field Access</i> .....	239
<i>Only Selected Fields Access</i> .....	241
<i>Record-Level Access</i> .....	241
<i>The Hierarchy of Access Rights</i> .....	243
<i>Database Cluster Access</i> .....	243
<i>About Read-Protected Fields</i> .....	243
TRANSFERRING DATABASE OWNERSHIP .....	245
RELATED CCL COMMANDS .....	245
<i>Show</i> .....	245
<i>Print</i> .....	246
<b>PART 5: APPENDIX AND INDEX .....</b>	<b>247</b>
<b>APPENDIX A .....</b>	<b>248</b>
GENERAL SETTINGS, LIMITS AND DEFAULTS .....	248
<i>Support for the Euro Currency Symbol</i> .....	248
<i>Searching for the Euro symbol</i> .....	248
<i>Support for the Chinese character set GBK</i> .....	248
<i>Limit to TRIPclassic CCL Command Length</i> .....	248
<i>No Limits to Database and Index File Sizes</i> .....	249
<i>Limit to the Number of Search Sets</i> .....	249
<i>Limit to the Number of Open Databases</i> .....	249
<i>Defaults for the DEFINE command</i> .....	249
<i>TRIPserver Crash Handling (Windows only)</i> .....	250
<b>APPENDIX B .....</b>	<b>251</b>
OBTAINING VERSION AND LICENSE INFORMATION .....	251
<i>TRIPmanager mmc Version Information</i> .....	251
<i>TRIPsystem Version Information</i> .....	252
<i>TRIP Product License Information</i> .....	252
<i>Updating a TRIP Product License Key</i> .....	252
TRIP USER ACCOUNT VALIDATION METHODS .....	254
<i>Overview</i> .....	254
<i>LDAP</i> .....	254
<i>Local System Validation</i> .....	254
<i>TRIP Standalone Usernames</i> .....	255
CONNECTING TO TRIP SERVERS .....	256
<i>Server Connection Overview</i> .....	256
<i>Creating a Server Connection</i> .....	257
<i>Specifying Credentials</i> .....	259
<i>Logging into the New Server Connection</i> .....	261
TRIP GRIDS .....	262
<i>Introduction to TRIP grid computing</i> .....	262
<i>Creating a Grid</i> .....	264
<i>Creating a Grid Cluster</i> .....	264
<i>Creating a Grid Replica Set</i> .....	265
<i>Publishing to a Replica Set</i> .....	266
<i>Publishing to a Grid Cluster</i> .....	266
<i>Grid Authentication</i> .....	266
<i>Advanced Grid Properties</i> .....	267

## Table of Contents

CLASSIFICATION SCHEMES.....	269
<i>Introduction to Classification Schemes</i> .....	269
SCOPE SEARCH FACILITY .....	273
<i>The new Scope Search facility</i> .....	273
<i>Scope Search Example</i> .....	273
<b>APPENDIX C: .....</b>	<b>276</b>
TRIP PROGRAMMING .....	276
TForm .....	276
<i>Control Strings</i> .....	276
<i>Text Strings</i> .....	277
<i>Record, Record Part, Field and Subfield Markers</i> .....	277
<i>Adding Records with TForm</i> .....	279
<i>Updating Records with TForm</i> .....	282
<i>Data Type STring and the Length Marker</i> .....	283
<i>Copying Records Using Print TForm</i> .....	283
APPLICATION SOFTWARE EXITS (ASEs) .....	284
<i>Summary</i> .....	284
<i>The Format of an ASE Routine</i> .....	286
<i>Linking ASE Routines to TRIP</i> .....	287
<i>Debugging ASE routines</i> .....	289
CCL ASEs .....	289
Output Format ASEs .....	290
TForm Load ASEs .....	292
Index ASEs .....	297
Data Entry ASEs (TRIPclassic only) .....	301
Search Form ASEs (TRIPclassic only) .....	305
TRIPsystem Callback Functions for ASE Routines .....	305
TRIPclassic Callback Functions for ASE Routines .....	305
TRIP API Reference Guide .....	306
<b>LIST OF FIGURES AND TABLES .....</b>	<b>307</b>
FIGURES .....	307
TABLES .....	308
<b>INDEX.....</b>	<b>310</b>



## About This Guide

### Scope and Assumptions

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

This guide does not cover the installation and set up of the TRIPmanager mmc plug-in. For more information on this subject, consult the TRIPmanager Installation Guide.

Furthermore, this guide assumes that the administrator performing the TRIPsystem management has sufficient access rights to manage the TRIPsystem installation in question and possesses a valid username and password to log into that system.

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

Further guidance can be found in the contextual help systems provided with the mmc and also with the TRIPmanager plug-in.

The help for TRIPmanager can also be reached directly via the TRIPmmc.chm help file in the TRIPmanager installation directory.

### The TRIP Documentation Library

Other members of the TRIP documentation library include the Installation Guides, Release Notes, Change Histories, TRIPmanager User Guide, TRIPclassic Administration guide, TRIPclassic User Guide, CCL Command Reference and the TRIPtoolkit.chm help file.

If you are not already familiar with searching and/or CCL (TRIP's Common Command Language), we recommend that you read the TRIPmanager User Guide first, placing special emphasis on the basic commands Find, Display, Show, Print, DEfine, List, BASE, DElete, SStatus, Run, SAve and Help. Additional information on these and other commands is available in the CCL Command Reference.

It is also important to understand the elements of data entry before attempting database construction. If necessary, review Chapter Nine of the TRIPclassic User Guide for data entry basics before proceeding.

## The Structure of this Guide

This guide is divided into six main parts, Database Administration, Forms, Batch Update, Database Security and the Appendices:

- Part I: Database Administration (Chapters One through Four) contains TRIP fundamentals, as well as everything you will need to begin creating and using TRIP applications, databases, thesauri and usage statistics.
- Part II: Forms (Chapters Five through Seven) covers using TRIPmanager to create, maintain and use TRIPclassic data entry forms, reports and search forms.
- Part III: Batch Update (Chapters Eight and Nine) includes global updating and loading and indexing of data.
- Part IV: Database Security (Chapters Ten and Eleven) discusses database access and the administration of user rights.
- Part V: The appendices which, in order, discuss:

### Appendix A

- General Settings, Limits and Defaults

### Appendix B

- Obtaining TRIP Version Information
- TRIP User Account Validation Methods
- Connecting to TRIP Servers
- TRIP Grids

### Appendix C

- TFORM (The Trip output FORMat)
- TRIP ASEs (Application Software Exits)

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

**NOTE:** The chapter about environment and logical names have from TRIP version 8.0 been moved into its own document, "TRIPsystem Environment".

## Conventions Used in this Guide

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

<i>italic</i>	used to indicate variables such as fieldtype or databasename, and to emphasize important terms and concepts
<b>bold</b>	used to indicate anything that TRIP recognizes or can interpret and act upon, such as the things mentioned above (<Tab>, CCL Search, DEfine, Alice, and wonderland)
lower case	used for terms and variables where variables are also italic
UPPER CASE	used for proper names such as the database ALICE
Courier	Courier Font is used to indicate examples containing specific text which you are to type in
< >	chevrons—used to indicate key(s) on the keyboard such as <Tab> or <Enter>
< >	space character
<CR>	carriage return
<LF>	line feed
<NL>	new line
<FF>	form feed
" "	messages from TRIP

In examples of CCL order syntax, square brackets [ ] indicate an optional construct, braces { } enclose option lists, a vertical bar [ | ] separates *exclusive* alternatives, and the ellipsis [...] designates a repeating construct.

## TRIP Naming Conventions

TRIP's naming requirements are presented in the following table:

Category	Content Alpha-numeric?	Length <sup>1</sup>	First Letter Alpha-betic?	Allowable Punctuation
File	Yes	128	Yes	Underscore
Database	Yes	64 (16)	Yes	Underscore
Field	Yes	64 (16)	No	Underscore
Procedure	Yes	64 (16)	No	Underscore
Output Format	Yes	64 (16)	No	Underscore
Entry Form	Yes	64 (16)	No	Underscore
Search Form	Yes	64 (16)	No	Underscore
Group	Yes	128 (32)	No	Underscore
User	Yes	128 (32)	No	Underscore
Password	Yes	128 (32)	No	Underscore

**Table 0–1 TRIP naming conventions**

<sup>1</sup> maximum length in characters (including file paths where applicable). The values in parentheses apply to programs using the pre-8.0 database design APIs (such as TRIPclassic).

## TRIP Logical Names

Throughout this guide, TRIPsystem, TRIPserver and TRIPclassic internal environment variables, normally defined in the [Privileged] and [Non Privileged] sections of the `tdbs.conf` file, are referred to as '*logical names*'; this has been done deliberately to avoid confusion with the identically named Windows and UNIX environment variables. Where the term '*environment variable*' is used, it refers to those variables defined in a Windows or UNIX user's environment.

*Note:*

*Using the `tdbs.conf` file is the recommended method for TRIP internal environment variable (logical name) creation.*

## Part 1:

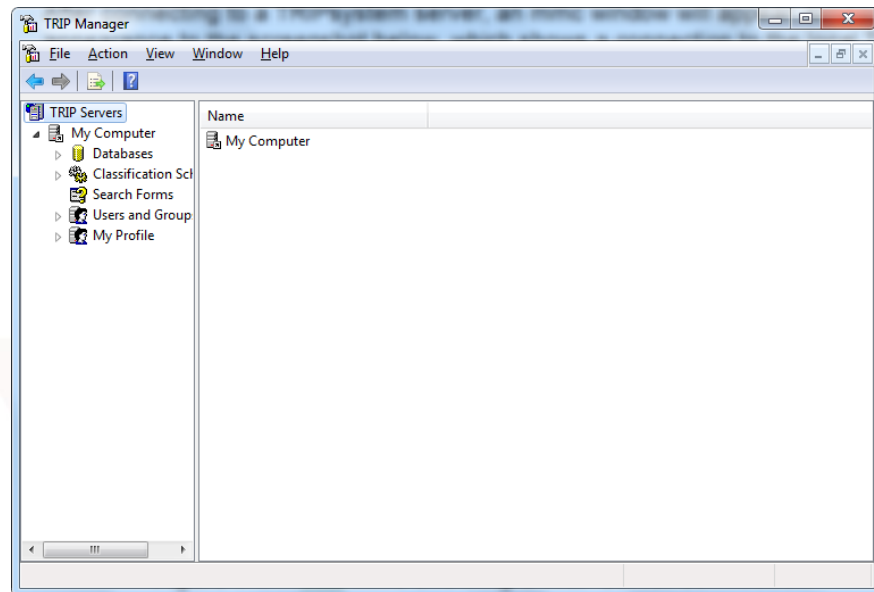
## Database Administration



## Chapter 1: Fundamentals

### Navigation within TRIPmanager

After connecting to a TRIPsystem server, the mmc window will appear, similar in appearance to the screenshot below, which shows a connection to the local TRIP server on a Microsoft Windows 7 installation:



**Figure 1-1 The mmc showing a TRIP server connection**

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

- Databases All databases accessible to use administrator accessing the server in question
- Search forms All search forms accessible to the administrator accessing the server in question
- Users and Groups All search forms accessible to use administrator accessing the server in question, assuming the username is granted user manager rights
- My Profile The profile belonging to the currently logged on user

and will be covered in detail later in this guide.

Throughout this guide, reference is only made to the 'Action' menu. Regular users of the mmc will, no doubt, already be aware that menu items within the mmc are contextual, as in most Windows applications; this is also the case with TRIPmanager, hence it is not always necessary to actually use the physical 'Action' menu on the mmc's menu bar. Therefore, where clarity is paramount, screenshots may actually show contextual menus rather than the 'Action' menu.

## TRIP System Basics

### Introduction

Three of the most popular types of data models are the flat file, the relational database and the full-text database management systems.

The first and simplest, the flat file system, is commonly used to store and manipulate large quantities of relatively unstructured and non-mission critical data where a return on any time investment made in data organisation is not expected. Uses might include an in-house corporate telephone listing, or a home music library catalogue kept in a spreadsheet.

The second, the relational database management system (RDMS), is useful where data exists in small, discreet units that lend themselves to rigorous organisation. Systems for inventory control and personnel management often use relational databases for data storage and manipulation.

The third, the full-text database management system or text database system (TDBS), is invaluable in the handling of large quantities of highly important, rather unstructured, data that must be searched extremely rapidly. Areas which find the TDBS of use are document management, standard operating procedures and the management of scientific data such as seismic exploration information. These database management systems, of which TRIP is representative, are also adept at object storage, including such data types as photographs, video images, voice imprints and hypertext.

These three data models are discussed in depth in the following section.

### Data Models

#### Flat Files

Defining a flat file as a data model is perhaps an exaggeration. However, many commercial applications take advantage of the relative ease of use that flat files offer, even though their search capabilities are quite limited.

Employee	Telephone Number
Fred Jones	(201) 555-1234
Ben Smith	(201) 555-8192
Ed Wedge	(201) 555-9999

**Table 1–2 Sample flat file table**

Typically, the only operators offered by such a 'search engine' are arithmetic (e.g. equal to, greater than, less than etc.), and the search mechanism varies from sequential scan, through indexed sequential scan to binary search (if the data in the file is sorted).

#### Relational Database Management Systems

Relational data models call for data to be organized in fixed-sized tables of related information, which are then 'joined' during a search to provide the flexibility required to retrieve meaningful results.

In the example below, two tables hold non-repetitive employee information (i.e. there is only one place in the database where the value of '10' is equated to the value 'Sales').

RDBMS data table EMPLOYEES

Employee	Title	EmployeeNo	DeptNo
Fred Jones	Clerk	1268	20
Ben Smith	Salesman	7582	10
Ed Wedge	Salesman	7654	10

✕ RDBMS data table DEPARTMENT

DeptNo	DeptName
10	Sales
20	Administration

**Table 1–3 Sample relational database tables**

Using an SQL (Structured Query Language) statement to extract all employees in the Sales department, the tables are joined using the field 'DeptNo':

```
SELECT Employee
FROM EMPLOYEES, DEPARTMENT
WHERE DEPARTMENT.DeptName="Sales"
AND EMPLOYEES.DeptNo=DEPARTMENT.DeptNo;
```

When designing such a data model, considerable effort is typically expended in constructing the various tables to ensure that data does not become redundant. This process is referred to as data normalization.

Also, much thought must be given to constructing the index for these tables so that, for instance, the join between the two 'DeptNo' columns can be performed as rapidly as possible. Without this extra effort, searches will complete extremely slowly.

### Full Text Database Management Systems

In contrast to other database models, the full text model calls for complete indexing of all possible database content. This frees database designers to spend more time on user interface issues such as form design and appearance, rather than on maximizing data model efficiency. As a result, full text applications tend to focus on large bodies of often natural language text, including books, documents and log files, rather than on small, discreet units of information which are more suited to relational database management systems.



## Indexing the Data

The fragment index is a hitherto unique feature of the TRIP database system which provides significant performance increases when searching for truncated terms. For instance, in the following table, the fragment index is used to locate terms, which in turn are used to locate content within the database itself. The example the CCL order:

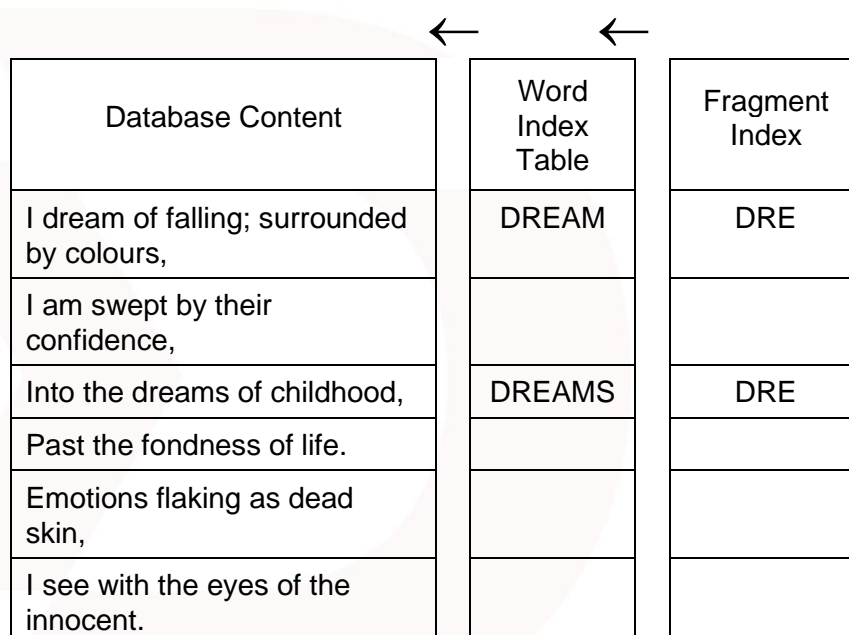
`Find $DRE$`

finds any term in the database which contains 'dre', in this case, 'dream' and 'dreams'.

As full text systems maintain a complete index, vocabulary listings are also a common feature. For example:

`Display $DRE$`

will return a list of all terms in the index which contain 'dre'.



Database Content	Word Index Table	Fragment Index
I dream of falling; surrounded by colours,	DREAM	DRE
I am swept by their confidence,		
Into the dreams of childhood,	DREAMS	DRE
Past the fondness of life.		
Emotions flaking as dead skin,		
I see with the eyes of the innocent.		

**Table 1–4 Sample full-text database table**

## Data Organisation

TRIP is a database system which has been specifically designed and implemented to handle the large amounts of variable length data, which is typical of free text applications. Free text is used here to mean natural language text, as found in books, letters, reports, log files, etc.

It is the unpredictable length of the data strings encountered in such applications which accounts for the main technical difficulties in designing and implementing a system to handle such data efficiently. Most conventional DBM (database management) systems deal mainly with fixed length blocks of data, and possess a very limited capacity for manipulating variable length text data. These field length fluctuations influence both the file structures and data access methods adopted during TRIP system design, and it is here that TRIP shows itself uniquely well placed for building this type of application.

The choice of data which TRIP has been implemented to handle most efficiently determines, to some extent, the contents of typical fields within the

system. Free text documents are, for example, normally broken into chapters, sections and paragraphs. Paragraphs are further subdivided into sentences, and sentences into words. In terms of fields, the most natural choice might be the collection of paragraphs into sections or chapters. Thus fields in TRIP which contain textual data might typically contain a number of paragraphs.

Within TRIP, the record level of organisation can be equated to a document, and the database may correspond to a collection of related documents.

Meta-record structures are also available, in which the head record contains information common to a number of sub-entities. A meta-record can be used to describe a collection of articles in a periodical, the head record containing such information as journal title, publisher, etc. and each part containing specifics regarding individual articles such as author, text and references cited.

A database might alternatively consist of a number of product descriptions. Each product would have its own record within the database and each record could consist of fields for the product name, product number, product description and date of introduction. If this database employed meta-record structure, record parts might then contain serial number, production run number, alterations from the basic model, etc.

## TRIP Field Types

Although TRIP was designed specifically for the efficient manipulation of free text, most documents have auxiliary information associated with them which are not free text, such as dates, times, numbers, authors, publishers etc. To accommodate varying data formats, TRIP supports seven data types, **TExt**, **PPhrase**, **DAta**, **TIme**, **NUmber**, **INteger** and **STring**, as described below.

**TExt** stores free text in sentences and paragraphs. There can be any number of paragraphs within a **TExt** field, which in turn may have any number of sentences of any length.

The position of every word in the text is noted in the appropriate file when the records are indexed, including the number of the paragraph within the text field, the number of the sentence within the paragraph, and the number of the word within the sentence.

**PPhrase** usually contains short text elements, e.g. names, addresses, identifying numbers or product codes. Each individual phrase constitutes one subfield. There can be any number of subfields within a **PPhrase** field and phrase fields may contain any number of characters, however while all words in the entire phrase will be word indexed, unlike in a **TExt** field, the whole phrase index will only use the first 255 (normalised) chars

*Note:*

*In this context, 'normalising' means first replacing all blank equivalents with blanks, then removing all leading and trailing blanks, as well as compressing all sequences of blanks to just a single blank.*

*Example (where ' ' represents a blank or space character):*

*The phrase " Tarzan, Jane and Cheeta like bananas!!!" will be normalized to "TARZAN JANE AND CHEETA LIKE BANANAS").*

When records which contain **PHrase** fields are indexed, the phrase along with all subfield contents (as well as each individual word) is noted in a TRIP index file along with its position (the number of the subfield within the field and the number of the word within the subfield).

*Note:*

*A phrase field can be any length but the index term for the complete phrase will be maximum 255 chars (normalized as described above). However, each single word of a phrase of any size will be indexed. The limit of 255 chars in the index only affects the whole phrase.*

**NUmber** holds double precision signed 64-bit real numbers.

*Note:*

*A database with **NUmber** values larger than would fit into a signed 32-bit floating point cannot be used with older versions of TRIPsystem than 8.0 without risking system stability.*

**INteger** holds signed 64-bit integer values.

For greater accuracy, use the data type **INteger** instead of **NUmber** whenever possible.

*Note:*

*A database with **INteger** values larger than would fit into a signed 32-bit integer cannot be used with older versions of TRIPsystem than 8.0 without risking system stability.*

**DAte** stores dates, primarily of the form year-month-day (e.g. 1985-04-20 or 85.04.20). A year only (1985 or 85) or a year and month only (1985-04 or 85-04) may also be used when entering data or searching

This is the standard date form, but other date forms are available. See the 'Date Form' section in Chapter Ten of the TRIPclassic User Guide entitled 'User Administration' for more information.

**Time** holds the time of day, expressed in 24-hour nomenclature of hours, minutes, and seconds (e.g. 11:04:02 or 11.04.02). The hour only, or the hour and minute only may also be given when entering data or searching. It is indexed in the same manner as **NUmber**.

**STring** contains a string of characters of any kind, i.e. images, video, voice etc. Data of type **STring** cannot be indexed.

The field type determines some aspects of the manner in which the system accesses data held within a field, as well as its indexing. It also determines to some extent the information that can be entered into that field. For instance, in **DAte** and **Time** fields there is an implicit validation employed that ensures that the data entered can be interpreted as a date or time.

**Text** data is organized into paragraphs and sentences. **PHrase**, **DAte**, **Time**, **NUmber** and **INteger** data types may be divided into an unlimited number of subfields, which can then be used for range searching within the database. **STring** fields are stored within the database, but are not indexed, and so cannot be retrieved using TRIP's query language, CCL.

## The CONTROL Database

A database system needs some method of tracking all of its parts or components, which in TRIP include users, user groups, databases, clusters, thesauri, reports, data entry forms, search forms, procedures and macros. This is done by way of the system data dictionary, a database known as CONTROL which contains definitions of the data structures currently within the system. Each definition within CONTROL occupies a separate dictionary entry.

The contents of the CONTROL database are illustrated schematically below.

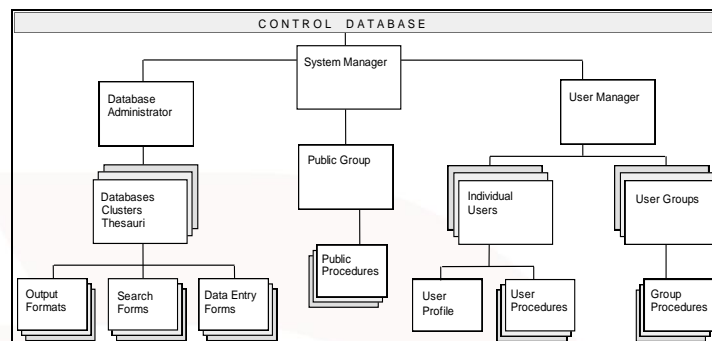


Figure 1–2 The CONTROL database

## TRIP Manager Privileges

Management responsibilities within TRIP are divided between three classes of administrator:

### System Manager:

the prime user within TRIP; assigns selected users database administrator or user manager privilege and administers the public group (all users).

### User Manager:

creates and maintains individual users and user groups, as well as the procedures and macros which are private to these groups.

### Database Administrator:

(also known as file manager) creates and modifies databases with their associated reports, etc., and grants other users and user groups access to the data within these databases.

## TRIP Database Basics

### Records

The data contained in a TRIP database is organized in terms of records, each record consisting of a collection of fields. A record can have any number of fields, not all of which need be filled (empty fields do not impose any storage overhead).

Each record within a database is assigned a unique record number on entry into the database, which can be used when accessing data within the database. A record may also be assigned with a unique record name.

In some applications, a record may exist as a two-level tree structure called a meta- or composite record, composed of a head record and any number of part records. In this arrangement some of the fields within the record are shared by all of the part records, and are collectively known as the head record. The contents of the remaining fields are unique to the record entity, and together constitute one part record. If head and part records have been included in the design of any particular database, each field is by definition either a head or a part field for that database.

The head record, which includes the head fields, is described by the contents of these fields and generally contains information which is relevant to all its part records. The part records (each of which holds one or more of the part fields) are described by the contents of those fields and usually contain information which is applicable to that part record only. Head records with part records are illustrated in the following figure.

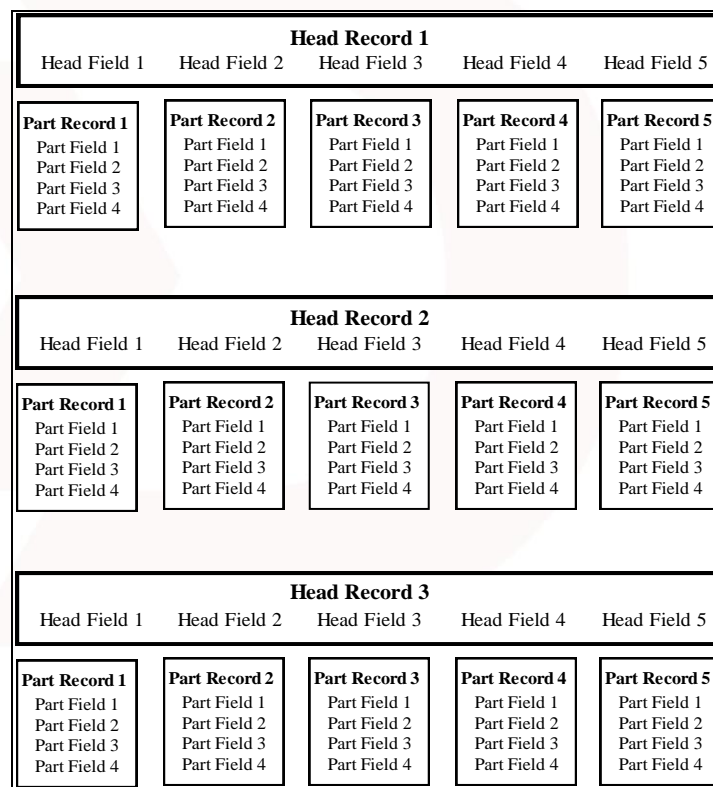


Figure 1–3 Head and part records in a database

Using the demonstration database Carroll as an example, head records made up of head fields now contain all of the chapter information contained in two books by Lewis Carroll—number and heading, the list of persons performing actions in the text and the title of the book from which the text was taken. The part fields within the part records hold the page information, and include the speakers in the text as well as all of the text fields.

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
<b>Part Record 1</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 2</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 3</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 4</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 5</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4

Head Record 1			
Book	Chapter	Chaptnr	Person
<b>Part Record 1</b> Speaker Txt Verse Txt2	<b>Part Record 2</b> Speaker Txt Verse Txt2	<b>Part Record 3</b> Speaker Txt Verse Txt2	<b>Part Record 4</b> Speaker Txt Verse Txt2

Head Record 2			
Book	Chapter	Chaptnr	Person
<b>Part Record 1</b> Speaker Txt Verse Txt2	<b>Part Record 2</b> Speaker Txt Verse Txt2	<b>Part Record 3</b> Speaker Txt Verse Txt2	<b>Part Record 4</b> Speaker Txt Verse Txt2

Figure 1–4 Carroll's head/part record structure

Each of the twenty-four main (chapter) records in Carroll has from one to thirty-seven part (page) records clustered beneath it.

The following figures illustrate head and part record terminology.

- The head record, containing all of the head fields.

Figure 1–5 A head record

- The part record, consisting of one set of part fields.

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
<b>Part Record 1</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 2</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 3</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 4</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 5</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4

Figure 1–6 A part record

- Record entities 1, 2, 3 etc. represent the union (or sum) of head record 1 and part record 1, head record 1 and part record 2, head record 1 and part record 3, etc.

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
<b>Part Record 1</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 2</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 3</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 4</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4	<b>Part Record 5</b> Part Field 1 Part Field 2 Part Field 3 Part Field 4

Figure 1–7 A record entity

- A composite or meta-record is the union of the head record and all of its part records.

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
<b>Part Record 1</b>	<b>Part Record 2</b>	<b>Part Record 3</b>	<b>Part Record 4</b>	<b>Part Record 5</b>
Part Field 1	Part Field 1	Part Field 1	Part Field 1	Part Field 1
Part Field 2	Part Field 2	Part Field 2	Part Field 2	Part Field 2
Part Field 3	Part Field 3	Part Field 3	Part Field 3	Part Field 3
Part Field 4	Part Field 4	Part Field 4	Part Field 4	Part Field 4

Figure 1–8 A composite record

- Record components are the unit records (individual head and part records) in the database.

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
<b>Part Record 1</b>	<b>Part Record 2</b>	<b>Part Record 3</b>	<b>Part Record 4</b>	<b>Part Record 5</b>
Part Field 1	Part Field 1	Part Field 1	Part Field 1	Part Field 1
Part Field 2	Part Field 2	Part Field 2	Part Field 2	Part Field 2
Part Field 3	Part Field 3	Part Field 3	Part Field 3	Part Field 3
Part Field 4	Part Field 4	Part Field 4	Part Field 4	Part Field 4

Figure 1–9 Record components

## File Structures

TRIP employs an inverted file organisation, in which the contents of every field within the database can be indexed. Consequently, the contents of every field can be searched, and records can be retrieved on the basis of these searches.

A logical database within TRIP (one whose structure is governed by the nature of the information contained within it, rather than the properties of its storage media) consists of three separate physical files, the BAF (BAsic File), BIF (Base Index File) and VIF (Vocabulary Index File).

The BAF contains data, while the BIF and VIF are indexes to that data and are used during data retrieval. The two index files are hashed tables, in which any given term has a unique location.

### The BAsic File (BAF)

This file holds the database information itself. Within the BAF, the conceptual level records are broken into a number of internal level records. In particular, TExt fields within records are broken up and stored as individual paragraph records within the BAF. This limits needless indexing of large TExt fields in their entirety, since only paragraphs which have been modified since the last indexing are reindexed.

### The Base Index File (BIF)

This file is used to store positional information for terms in the BAF. During the indexing process, the records in the BAF are scanned, and each term is extracted separately. As each term is read, its position within the database is recorded.

### The Vocabulary Index File (VIF)

This file is in effect the index file for the TExt or PHrase fields which occur in the BIF. Indexing here involves dissecting each term in the BIF into single



(unigram), double (bigram) and triple (trigram) letter combinations, each of which then becomes a term in its own right and is posted in the VIF.

*Note:*

*It is important to understand this concept when using TRIP's FUZZy search capabilities. Refer to the TRIPmanager User Guide and CCL Command Reference for further information regarding FUZZ and Find FUZZ.*

### **The Session Index File (SIF)**

The SIF stores all the information required to restart a search session following an unscheduled disconnection, this includes search and print order histories, search language (English, Swedish etc.), open thesauri and any maximums, minimums or mapping that have been defined. Sessions may also be intentionally saved to a SIF file using the CCL **STOP SAve** order: See the *CCL Command Reference* for more information.





## Chapter 2: Databases

### Notes on File Locations

The recommended method for administration of TRIP database physical paths is to create logical names in the `tdbs.conf` file, to be used as pointers to the locations of the database files.

This recommendation is to ease administration, as any later changes to file locations will only necessitate the altering of a single logical name, rather than having to alter many individual database designs.

In order to encourage this behaviour, the TRIPmanager Database creation wizard has been designed to offer a drop-down selection box containing only those logical names found in the `tdbs.conf` file; although it is possible to edit these for direct physical file paths at a later stage, if absolutely necessary.

Further details on how to create logical names are contained in the section entitled 'Physical File Locations'.

### Creating the Database

To create a database, activate the 'New Database Wizard' by clicking on the 'Action' menu item, 'New Database':

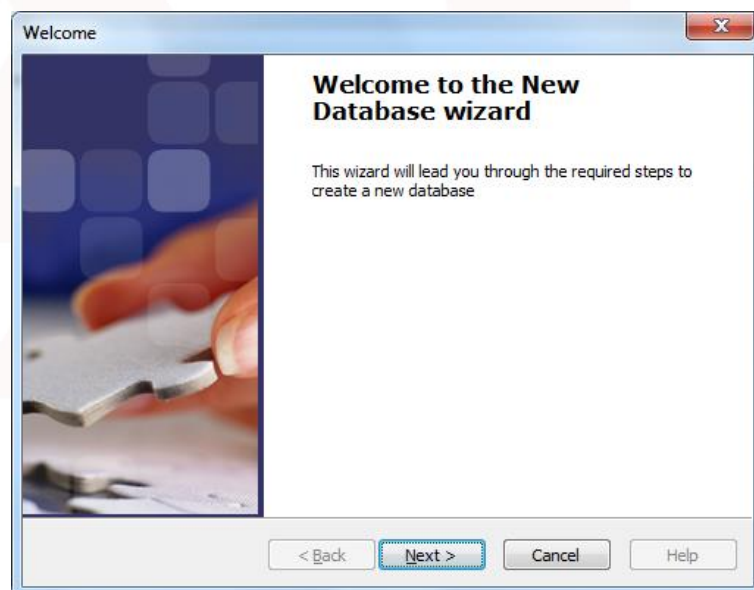


Figure 2–1 New Database Wizard

Clicking on the New Database Wizard's 'Next' button will take you to the general properties form:

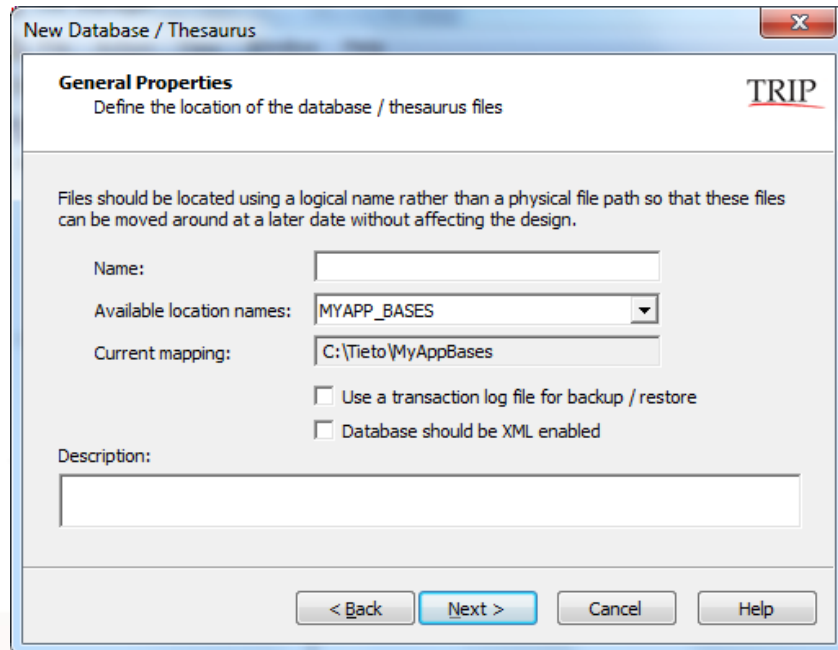


Figure 2–2 New Database General Properties

## General Database Properties

### Database Name

First you will need to enter the name of the database you wish to create.

A database name in TRIP may have, at most, 16 characters. The first character must be a letter; the others may be letters, digits, or underscores ( \_ ).

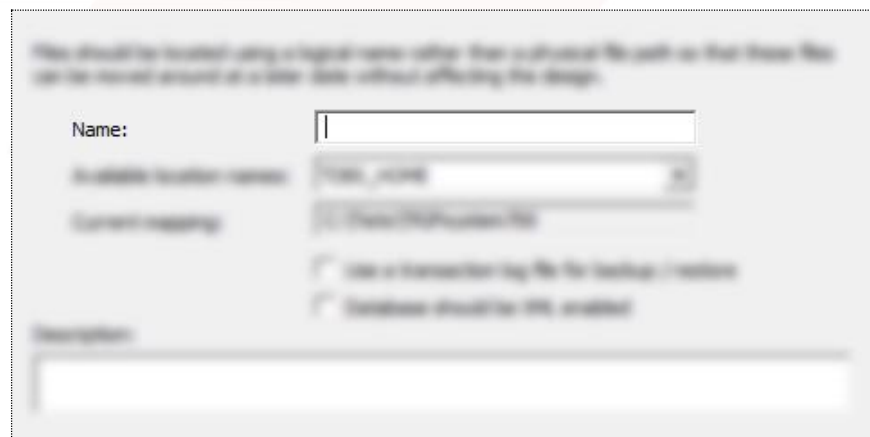


Figure 2–3 Database Name Entry Field

Type an appropriate (and preferably descriptive) name in the entry box beside 'Name:'.

## Physical File Locations

This portion of the form allows you to specify the location of the database files within the hosts file system. TRIP creates the physical database files when these specifications are saved, and the default file location (unless otherwise indicated) is the first item in the drop-down list.

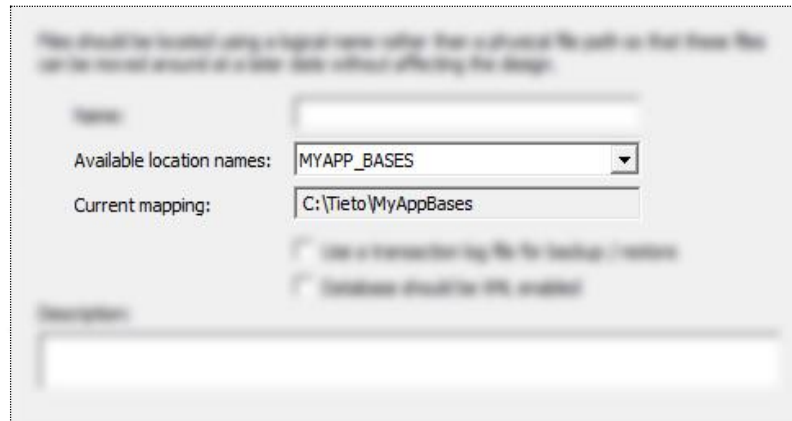


Figure 2–4 The Database File Location Selection Boxes

Clicking on the down arrow at the right-hand end of the 'Available locations names:' drop-down selection list, will display a list all available selections allowing one to be chosen. The greyed-out 'Current mapping:' box below the drop-down selection list, shows the physical path associated with the chosen logical name.

## Creating TRIP Logical Names

To create a new logical name, first quit any running instances of TRIPmanager (and/or TRIPclassic), then open the tdbb.conf file in your preferred text editor.

Next, anywhere in the [Non Privileged] section of the file, add a line of the format

```
Logical_Name=Physical_Path
```

where Logical\_Name is the name chosen for the new TRIP logical name and Physical\_Path is the actual operating system path to be mapped to the logical name; for example:

In Windows:

```
MyApp=C:\Users\Albert\TRIPapps
```

Would map the logical name MyApp to the path

```
C:\Users\Albert\TRIPapps
```

In UNIX:

```
MyApp=/home/sally/TRIPapps
```

Would map the logical name MyApp to the path /home/sally/TRIPapps

## Transaction Log

A transaction log records all changes made to the BAF, whether by data entry, global updating, or the loading of another TForm file. Should the most

recent BAF be damaged between backups, the log file and the BAF backup can be used to restore the BAF to its previous condition.

You can attach a transaction log file to the database by checking the 'Use a transaction log file for backup / restore' check-box shown below:

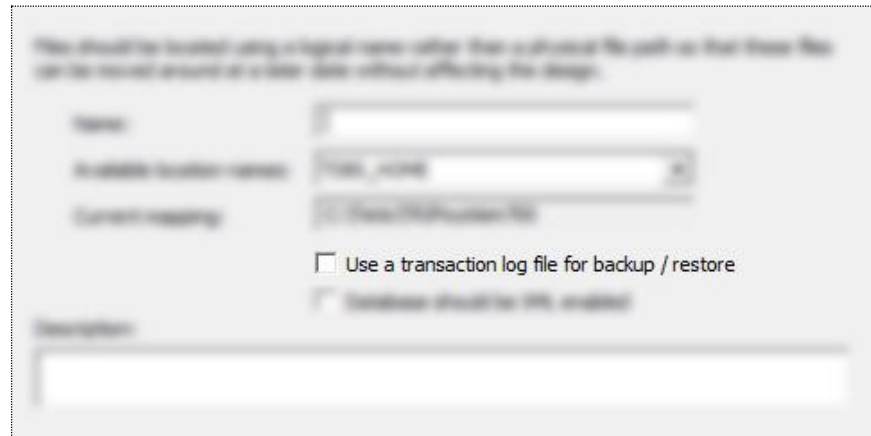


Figure 2–5 Transaction log selection

Remember that a log file is associated with a backup of a database. After creating a backup you should create a new empty version of its log file, for example:

in UNIX,

```
rm filename ↵  
touch filename ↵
```

in Windows,

```
del filename.log ↵  
type nul > filename.log ↵ (In a command window)
```

or

```
Remove-Item filename.log ↵  
New-Item filename.log -type file ↵ (In Windows Power  
Shell)
```

alternatively in Windows, you can delete the old log file, create a new text file, then rename it to the original name.

When attempting to reconstruct a damaged BAF, you should retrieve the last 'good' BAF from backup and then apply all subsequent log files to that BAF before backing up again. This is done using the database load/index menu (described in Chapter Nine of this manual), which specifies each log file in turn as the TForm file to be loaded into the BAF.

*Note:*

*When restoring the BAF, you should delete the transaction log name from the database design. If you do not, the old transaction log will be loaded as an input TForm file and written redundantly to the new transaction log. After restoration, you should replace the transaction log name to ensure the safety of your data.*

## XML Enabling the Database

Should it be necessary to create a database that will be used with TRIPxml, then check the check-box labelled 'database should be XML enabled'.

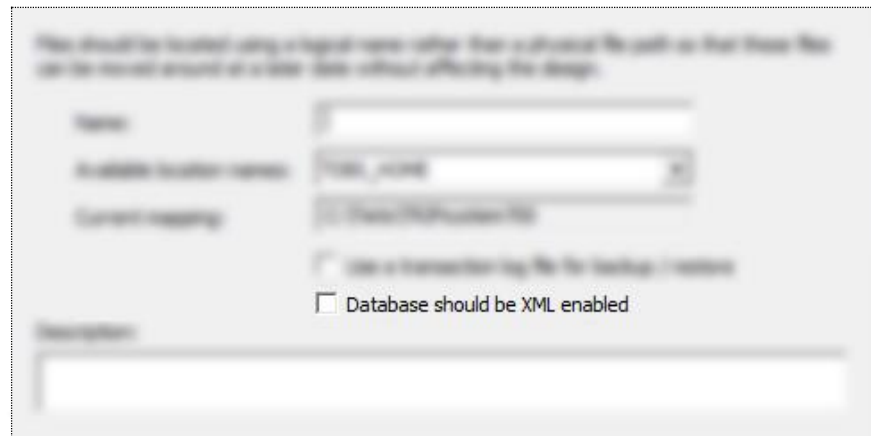
A screenshot of a software configuration window. At the top, there is a note: "This should be installed using a logical name rather than a physical file path so that those files can be moved around at a later date without affecting the design." Below this, there are several fields: "Name:" with an empty text box, "Available location names:" with a dropdown menu showing "TRIP\_XML", and "Current mapping:" with a dropdown menu showing "C:\Data\TRIP\Location file". There are two checkboxes: "Use a transaction log file for backup/restore" (unchecked) and "Database should be XML enabled" (unchecked). At the bottom, there is a "Description:" label followed by a large empty text area.

Figure 2–6 XML Enabling a Database

## Description of the Database

The last field listed on the General Properties form is the database Description field.

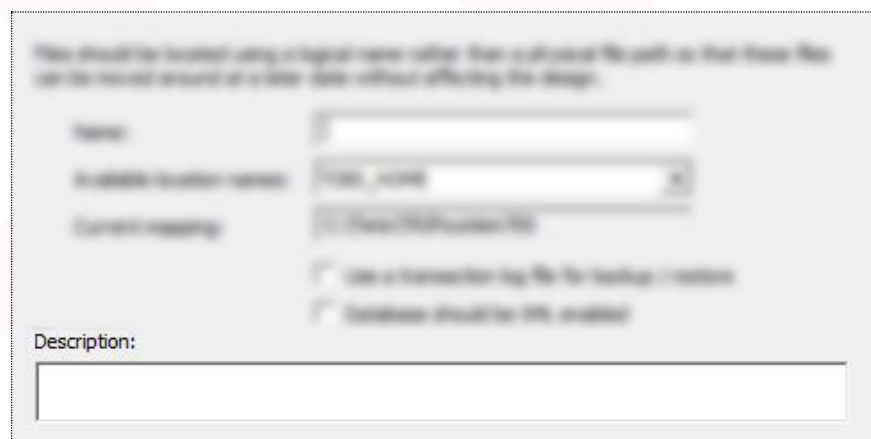
A screenshot of the same software configuration window as in Figure 2-6. The focus is on the "Description:" label and the large empty text area below it. The other fields and checkboxes are visible but not the primary focus.

Figure 2–7 The Database Description field

Here you can enter a description of the database to a maximum of 255 characters. This will be part of the information given when the database list is shown in the right-hand panel of the mmc, when the database properties are selected in the mmc, or when a CCL STATUS command is issued for that database.

## Saving the database design

Clicking on the 'Next' button at the bottom of the 'General Properties' form takes you to the completion page of the New Database Design Wizard:

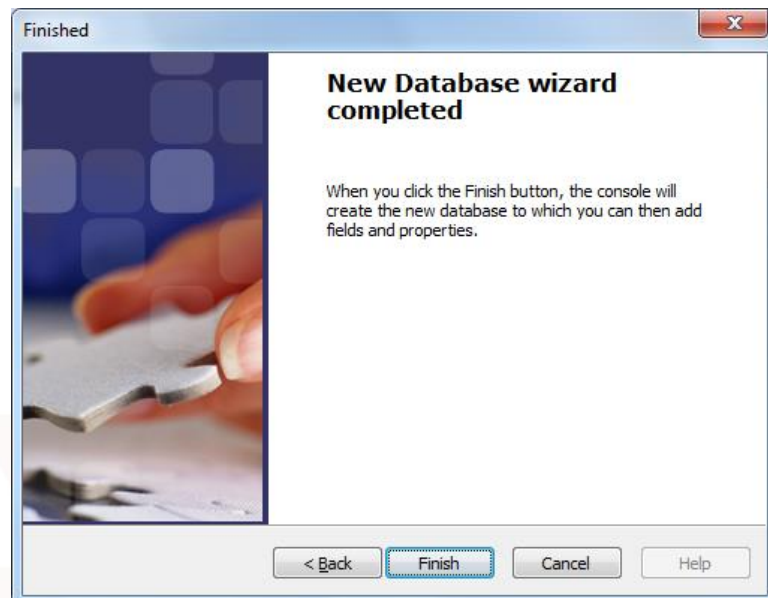


Figure 2–8 New Database Design Wizard Completion page

And clicking on the 'Finish' button saves the database design. This is confirmed with a pop-up dialogue box:

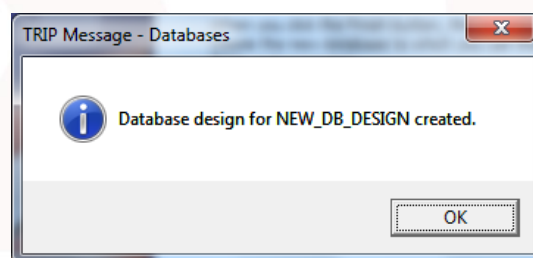


Figure 2–9 DB Creation Confirmation

Clicking on 'OK' opens a new dialogue box, asking if you wish to create the database's fields at this time:

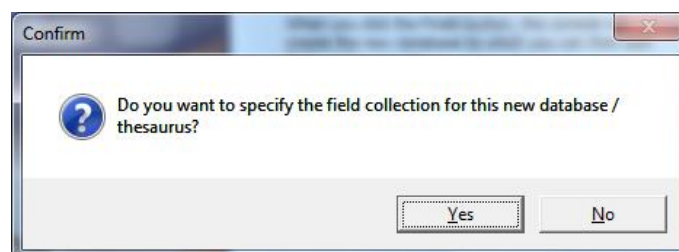


Figure 2–10 Specify Field Collection Query

If you do wish to create the fields now, click on 'Yes'. If you wish to create the fields later, click 'No'.

If you wish to go directly to database fields creation, turn to the section entitled, "Field Definition" on page 55 of this guide.

The next sections look at other database configuration parameters and how to modify them.

## Modifying Database Properties

### Database Properties (1) – General

The 'Create Database Wizard' creates a database with mostly default selections. Should it be necessary to alter these defaults, then it will be necessary to access the database properties sheet.

The properties for a selected database can be accessed via the 'Properties' entry on the Action menu. Selecting this entry displays a four tabbed form similar to that shown below:

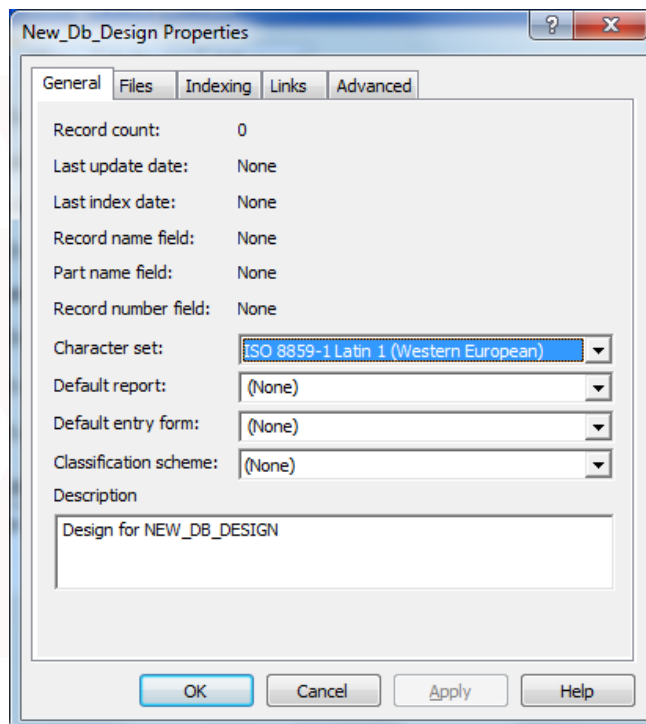


Figure 2–11 The Database General Properties Form

The first tab displayed is always the 'General' properties tab. This tab has six information fields (non-editable) and three user updatable fields. The non-editable information fields are:

- Record count: A count of the total number of records in the database
- Last update date: The date that the database was last updated
- Last index date: The date of the last index to be performed in the database
- Record name field: Which field, if any, is the Record Name field
- Part name field: Which field, if any, is the Part Record Name field
- Record number field: Which field, if any, is the Record Number field



The user definable fields are detailed in the following sections and are:

- Character Set:
- Default Report:
- Default Entry Form:
- Classification scheme:
- Description:

### Character Set:

This drop down box is used to select the default character set for the database.

*Note:*

*This value can only be changed in an empty (i.e. new) database*

### Default Report

The Default Report (formerly known as the Default Output Format) for the database, may be selected from the drop-down list of available reports.

If you do not provide the name of a default report, the system will show all output using its built-in default report 'Dump'. This report presents all non-empty fields contained within the database, headed by their field names. Unless a SORt or Show REVerse order has been given, records appear sorted in increasing record number order. Fields within records are output according to their field type, in this order:

PHrase, NUmber, INteger, DAte, TIme, TExt

The fields are listed in field number order within those field types.

The sample below was taken from the demonstration database Alice, using the CCL command

```
Show Format=dump ↵
```

In it, the PHrase fields chapter and person (field numbers two and three) are output first, then the INteger field chaptnr, and finally the TExt field txt:

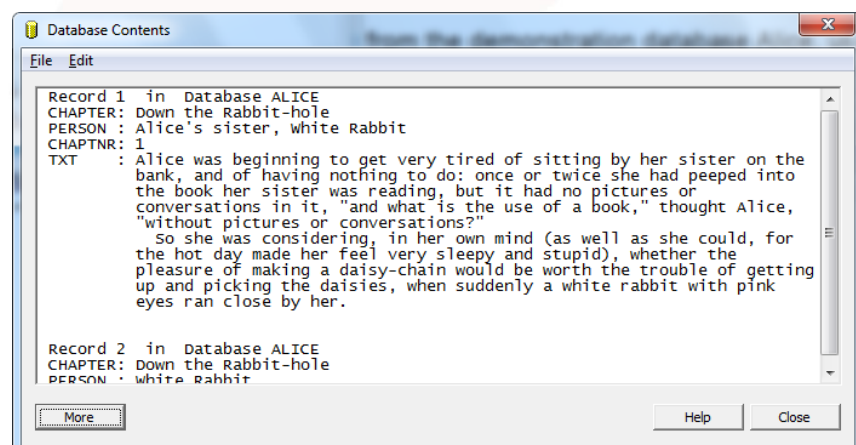


Figure 2–12 Sample SYSTEM default report, 'Dump'

Designating a default report allows the use of a simple Show command in CCL without a preceding DEfine Format statement. For example, rather than using the series



```
BASe alice ↵  
Find tweedle ↵  
DEfine Format=outputformatname ↵  
show ↵
```

a user may simply enter

```
BASe alice ↵  
Find tweedle ↵  
show ↵
```

after a search.

The DEfine Format statement is not needed if a report has been specified.

### Default Entry Form

The default Data Entry form for the database may be selected from a drop-down list of existing Entry Forms.

If the database is to be updated using interactive data entry (as opposed to global or TForm updating), you should name a default entry form, whether or not one or several different entry forms are to be used for the database.

If you have specified a default data entry form, TRIP will automatically provide the name of the default entry form after a user enters the name of that database.

Naming a default entry form also allows the CCL command

```
edit ↵
```

to be used without a preceding entry form name definition. For example, rather than using the command series

```
BASe alice ↵  
Find dee ↵  
DEfine EForm=entryformname ↵  
edit s=0 ↵
```

a user may simply enter

```
BASe alice ↵  
Find dee ↵  
edit s=0 ↵
```

after a search.

The DEfine EForm statement is not necessary if the default entry form has been included in the database design form above.

For greatest efficiency, the default form should be designed generically enough so that as large a population of write-privileged users as possible may have recourse to it. See the section entitled 'Creating a Data Entry Form' in Chapter Five of this manual for more information.

### Classification scheme

This drop down box can be used to attach a classification scheme to a database.

To do this, simply choose the name of any available classification scheme from the 'Classification scheme:' drop down box. The database will then need to be re-indexed to be classified, and any new or subsequently modified data will be processed for classification during normal indexing procedures.

*Note:*

*For more detail on how to create and manage classification schemes, see the relevant section in Appendix B and also the white paper entitled, "TRIP Document Classification", included with the TRIPsystem documentation.*

### Database Description

The database description, if one was entered during database creation, may be modified here or, if required, may be added here.

### Database Properties (2) – Files

Clicking on the 'Files' tab of the Properties will change the display to show the Database Files Property form. This form has two sections, one for collectively locating files by logical name and another for specifying individual file locations.

It is this the lower half of this form, 'File locations are specified individually', that should be used if you wish to locate files using full path names:

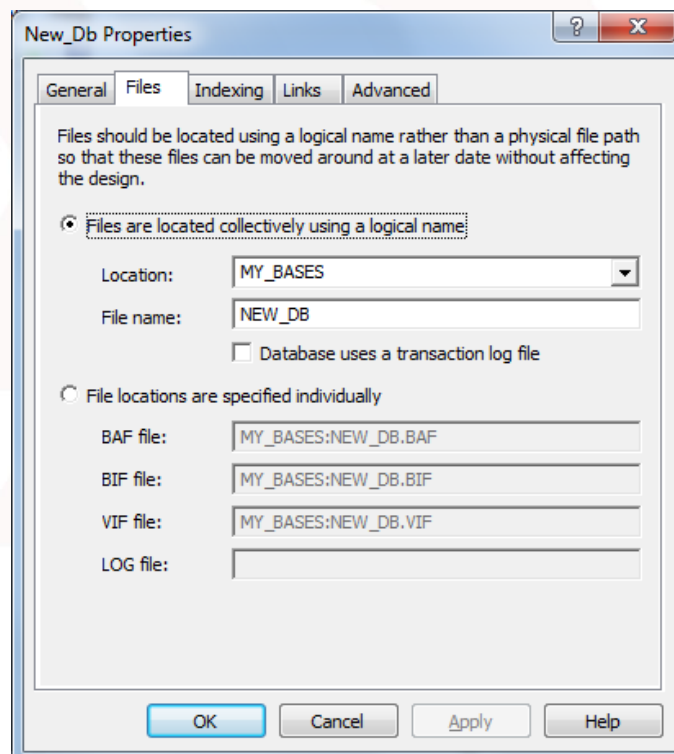


Figure 2–13 The Database Files Properties Form 1

#### Files are located collectively using a logical name

If the radio button is clicked for, 'Files are located collectively using a logical name', only three options will be available to the user:

- A drop-down selection box entitled 'Location'.
- A text entry box entitled, 'File Name'.
- A checkbox for selecting 'Database uses a transaction log'.

These are the same three entries that were made on the Create database Wizard's general Properties form and need no further explanation here.

Should you wish to alter these entries the methods described in creating a database also apply here.

### Individually specified File Locations

Clicking on the 'Files locations are specified individually' radio button will change the Database Files Property form display so that it becomes similar to below:

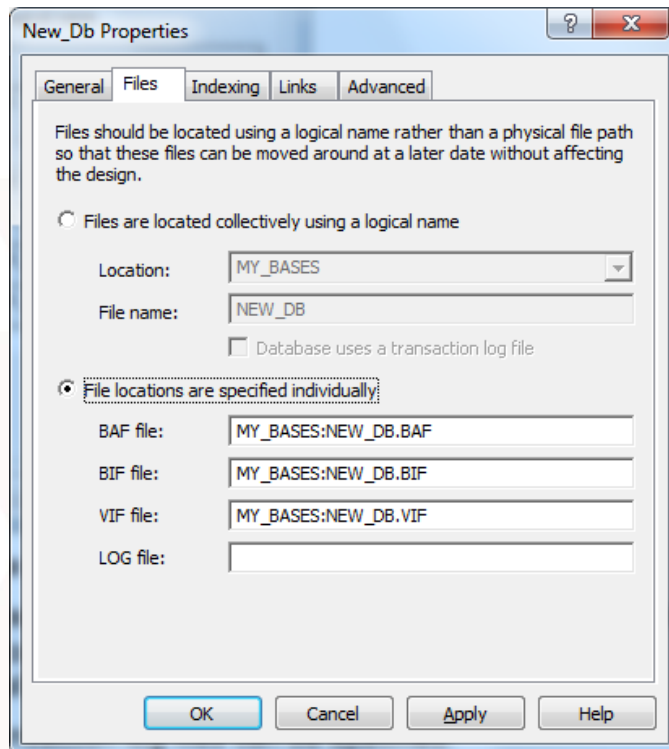


Figure 2–14 The Database Files Properties Form 2

As can be seen from Figure 2-13, each individual file location can now be altered and (though this is not recommended) should it be absolutely necessary, a physical file location could be entered for any or all of the files.

This is also where the location for transaction log files can be specified.

*Note:*

*Both a transaction log file name and file path, or file name and logical name representing a path, must be specified. If no log file location and/or name are entered, no transaction log will be created.*

## Database Properties (3) – Indexing

Clicking on the 'Indexing' tab of the Properties will change the display to show the Database Indexing Property form.

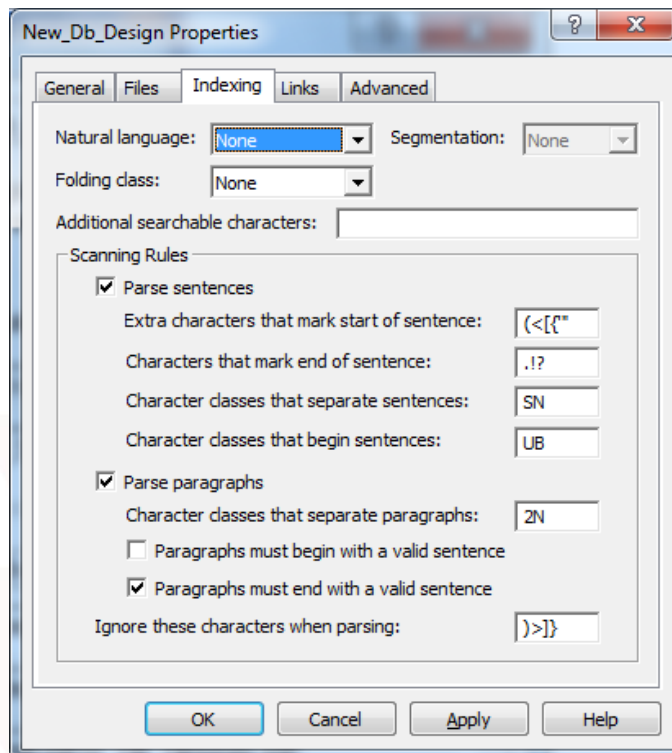


Figure 2–15 The Database Indexing Properties Form

The upper section of this form has drop-down selection boxes for 'Natural language', 'Folding class' and 'Segmentation' and a text entry box for specifying additional searchable characters, while in the lower section there are controls for specifying the rules used when creating database indexes. These subjects are covered in more detail in the following sections:

### Character handling

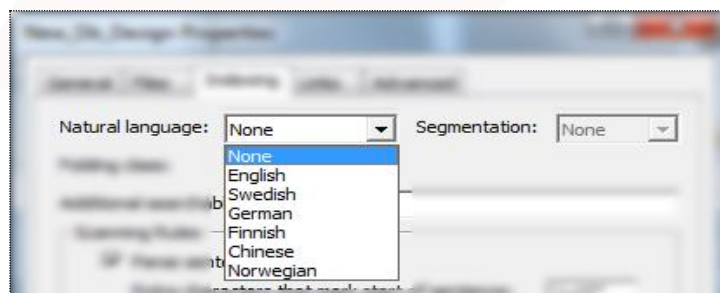


Figure 2–16 Natural Language Treatment selection box

The '*Natural Language*' drop-down selection box allows for the selection of a default language for use in indexing the selected database. The current alternatives are: '*None*' (default), '*English*', '*Swedish*', '*German*', '*Finnish*', '*Chinese*' and '*Norwegian*'.

*Notes:*

- *The language selected here will be used as the default for stemming searches (See the “CCL Command Reference – Display STEMming” section for more information.)*
- *If no language is selected here, the value defined by TDBS\_LANG will be used.*
- *As detailed in the next section, the ‘Segmentation’ selection box will only ever be activated when ‘Natural language’ is set to ‘Chinese’.*

### Chinese word segmentation

If the natural language chosen is set to Chinese, you can also choose to use smart work tokenization, which switches on a special algorithm that attempts to split Chinese character streams into words. If you index a database in Chinese without this option, each character in the character stream is treated as a separate word.

*Note:*

*If Chinese is stored in a database that is not setup for Chinese natural language processing, that data will be unsearchable.*

TRIP supports four different methods for Chinese Word Segmentation:

- **M – Maximum**  
This method selects the maximum length Chinese words from a string, based on a dictionary containing words of length 2-10 Chinese characters. This method also handles cross-ambiguities and continuous-cross-ambiguities of Chinese words correctly.
- **W – Word**  
This method is similar to method M and adds re-segmentation of all words longer than three Chinese characters.
- **A – All**  
This method segments every possible Chinese word as well as all single-character Chinese words.
- **N – None**  
This method indexes every single Chinese character as a word on its own.

### Folding class

Every text database system intended for use in more than one country must possess a sort method for multinational characters, which encompasses those characters not found in the system designer’s native language. An overview of these special characters follows:

A	C	E	I	N	O	S	U	Y
À		È	Ì		Ò		Ù	
Á		É	Í		Ó		Ú	Ý
					Ø			
Â		Ê	Î		Ô		Û	
Ã				Ñ	Õ			
Ä		Ë	Ï		Ö		Ü	
Å								
Æ					OE			
	Ç							
						ß		

**Table 2–1 Special characters**

The Folding Class selection box allows the database designer to specify how characters outside his or her native language will be treated during sorting and indexing, where one letter is regarded as having the same indexed value as another letter. Databases using different character folding methods cannot be searched simultaneously.

The character folding classes available are *English*, *Swedish*, *German*, *Finnish* and *Norwegian*.

The default is that no character folding is done; e.g. an é is a singular character and is separate from e, ä is not indexed as a, and so forth.

The folding class *English*, does not recognize diacritics or umlauts, so that é, è, ê (and so on) are folded onto e, ä onto a, ö onto o, etc.

The folding class *Swedish*, is identical to English except that å, ä, and ö are recognized as singular characters.

The folding class *German*, is identical to English except that ü and ö, ä and ß are recognised as singular characters.

The folding class *Finnish*, is identical to English except that ä and ö are recognised as singular characters.

The folding class *Norwegian*, is identical to English except that Æ, ü and ø are recognised as singular characters.

The ways in which the five main classes treat various characters are outlined below.

*Note:*

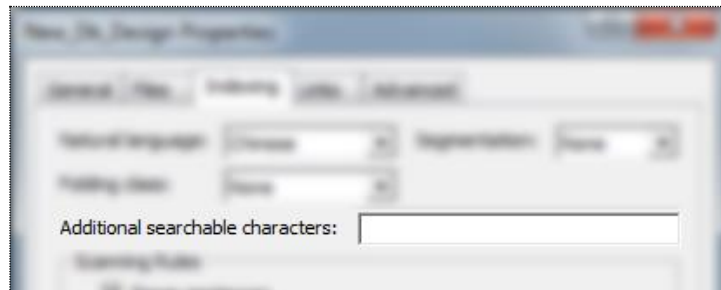
*CHinese is handled separately, using a different character set (GBK):*

Latin1 (MULTinational)	ENGLISH	SWEdish	GERman	NORwegian
À	A	A	A	A
Á	A	A	A	A
Â	A	A	A	A
Ã	A	A	A	A
Ä	A	Ä	Ä	Æ
Å	A	Å	Å	Å
Æ	A	Ä	Æ	Æ
Ç	C	C	C	C
È	E	E	E	E
É	E	E	E	E
Ê	E	E	E	E
Ë	E	E	E	E
Ì	I	I	I	I
Í	I	I	I	I
Î	I	I	I	I
Ï	I	I	I	I
Ñ	N	N	Ñ	Ñ
Ò	O	O	Ò	Ò
Ø	O	O	Ø	Ø
Ó	O	O	O	O
Ô	O	O	O	O
Õ	O	O	O	O
Ö	O	Ö	Ö	Ö
Œ	O	Œ	Œ	Œ
ß	S	S	ß	S
Ù	U	U	U	U
Ú	U	U	U	U
Û	U	U	U	U
Ü	U	U	Ü	Ü
Ý	Y	Y	Ý	Ý

**Table 2–2 The character folding classes**

### Additional searchable characters

The searchable characters in the text parts of a TRIP database are letters and digits by default. However, you may add to this set of characters by specifying extra searchable characters in the Additional searchable characters input box.



**Figure 2 - 1 Additional searchable characters input box**

Any character except the single ['] and double ["] quotes may be made searchable by typing the characters into the Searchable Special Characters design field without separators.

*Note:*

*You cannot combine multiple databases with disparate searchable character sets in a search.*

The characters used as truncation symbols, character masks, word masks, delineators, operators or sentence separator defaults should not be designated as searchable, since they have special functions in search orders. These symbols are listed below:

Symbol	Reserved Function
\$	truncation and masking
#	truncation and masking
&	masking
.	masking, sentence separator
!	truncation and masking, sentence separator
?	sentence separator
:	truncation and masking
( )	delineators
+	AND operator

**Table 2–3 Truncation, masking and special symbols**

The sentence separator defaults are included in the table above since during the execution of a word-string search order, TRIP searches for the given terms only within the same sentence (or subfield), unless the meaning of the space character is redefined by the CCL order, **Define SPACE**.

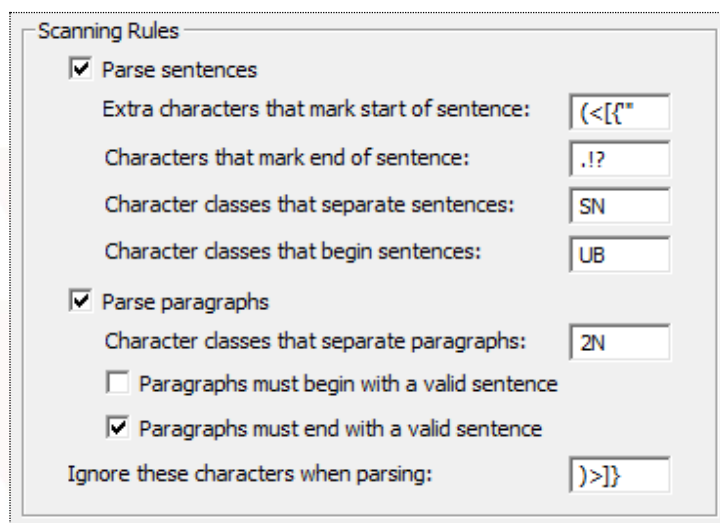


Characters not defined as searchable are treated as space characters, both during searching and indexing. For example, unless the hyphen [-] has been included in the searchable character set for a database, TRIP will interpret both 'on-line' and 'on line' as dual-word rather than single-word terms.

Refer to the *CCL Command Reference* for further information regarding **DEfine SPace**, truncation and masking.

### Scanning Rules

The scanning rules section of the database properties " form, permits the modification of TRIPsystem's scanning rules for indexing sentences and paragraphs.



The image shows a screenshot of the 'Scanning Rules' sub-form. It contains several settings for parsing sentences and paragraphs. The 'Parse sentences' section is checked, with default values for extra characters marking the start of a sentence ('(<[{'), characters marking the end of a sentence ('.!?'), character classes separating sentences ('SN'), and character classes beginning sentences ('UB'). The 'Parse paragraphs' section is also checked, with a default value for character classes separating paragraphs ('2N'). There are checkboxes for 'Paragraphs must begin with a valid sentence' (unchecked) and 'Paragraphs must end with a valid sentence' (checked). At the bottom, there is a field for 'Ignore these characters when parsing:' with the default value '}>}'.

**Figure 2 - 2 The Indexing Scanning Rules sub-form**

The system defaults for each category are presented as seen in Figure 2 - 2 above.

### Sentences and Paragraphs

Unless otherwise instructed, TRIP will separate text into paragraphs and sentences by defaulting to a set of predefined rules. The internal text separation rules can be altered by customizing the sentence and paragraph delimiters from the General Database Properties form, causing TRIP to separate the text into paragraphs and sentences accordingly.

### Character Classes

Ten character classes are available to facilitate paragraph and sentence recognition specification. Each character is by default assigned to only one of the following categories:

Char. Class	Content Description	Contents	Alter Definition?
L	Lower Case Letters	a b c ... z	No
U	Upper Case Letters	A B C ... Z	No
D	Digits	0 1 2 ... 9	No
S	Space Equivalents	ASCII values 0-32, minus Class N	No
N	New Line Equivalents	<LF> <VT> <FF>	No
H	Hyphen	-	No
R	Reset	Variable	Yes
B	Special Sentence Begin	(<[ { « "	Yes
I	Ignore	)>]}» and <CR>	Yes
E	Sentence End	.!?	Yes

**Table 2–4 The character classes**

Of these, L, U, D, S, N and H cannot be changed, i.e. no characters can be added to or removed from these classes.

Classes B, I and E may have character sets defined for each (default contents are shown in the preceding table).

Class N consists of the <LF> (Line Feed), <VT> (Vertical Tab) and <FF> (Form Feed).

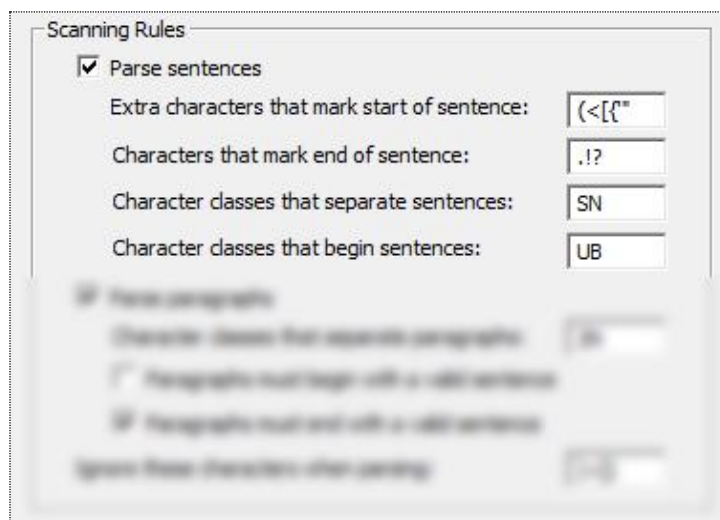
With the exception of the <CR> (Carriage Return), which is permanently assigned to Class I, Class S contains the space character (ASCII value 32) as well as all control characters (ASCII values 0-31) not belonging to Class N.

Class R contains those characters that do not belong to any of the other classes. Characters that are added to or removed from the B, I or E classes are automatically removed from or added to Class R respectively.

Classes L, U, D, H and B are known collectively as the 'Sentence Begin' classes, in which any characters from these classes can define the start of a new sentence.

### Defining a Sentence

The fields shown below allow you to define the beginning, the end and the separations between sentences.



The image shows a 'Scanning Rules' dialog box. The 'Parse sentences' checkbox is checked. The 'Extra characters that mark start of sentence' field contains '(<[{"'. The 'Characters that mark end of sentence' field contains '.!?''. The 'Character classes that separate sentences' field contains 'SN'. The 'Character classes that begin sentences' field contains 'UB'. Below these are several unchecked checkboxes: 'Parse paragraphs', 'Paragraphs must begin with a valid sentence', 'Paragraphs must end with a valid sentence', and 'Ignore these characters when parsing'.

**Figure 2 - 3 The Sentence Parsing fields**

The default values for sentence parsing are shown in Figure 2 -18 above. A more detailed description of these conditions follows.

### Parse sentences

Checking or clearing the 'Parse sentences' checkbox will respectively switch sentence separation on and off

If you choose to have sentence separation switched on and then attempt to perform a non-exact match PHrase search in CCL such as:

```
Find mad hatter ↵
```

the terms 'mad' and 'hatter' will not be found unless they appear as contiguous terms in the text and are in the same order as specified in the CCL command.

If you choose No, the terms can be split across what would normally be considered as several sentences, and TRIP will still find them.

If you choose Yes and your text contains errors (perhaps during import from OCR) such that extraneous characters have been accidentally inserted into one or more phrases (for example, a period [.] or other character from Class E), then these corrupted terms will not be found during straightforward CCL searching.

Using the 'mad hatter' example above, if 'mad' became modified to 'ma.', TRIP would consider 'ma.' and 'hatter' to reside in separate sentences, and would not find the term 'mad hatter' wherever this has occurred.

Choosing No for Sentence Separation may be advantageous in instances such as the one above, where you may be unsure of the integrity of your incoming data. Since TRIP does not parse for sentences, this may also be faster; however, without Sentence Separation you will be unable to perform CCL proximity searching with constructs such as AND.S.

If you choose No, import data to your database and then change your separation selection to Yes, you must then rebuild the database (that is, print the database into TForm, delete the database and reload it from TForm), since TRIP will not reparse the data that was already loaded (parsing occurs only on the instream, or incoming data).

### Extra characters that mark start of sentence

This option specifies which character(s) belong to Class B. If characters are removed from Class B, and not placed in Classes I or E, they will be moved to Class R. This option cannot exceed thirty-two characters. The default system values are ( < [ { « ' "

### Characters that mark end of sentence

This option, represented by Class E, specifies which character(s) will identify the end of a sentence. If any characters are removed from this class and not placed in Classes I or B, they will be moved to Class R. This option must contain between one and sixteen characters, unless you select No for Sentence Separator (see below). The default system values are .!?

### Characters classes that separate sentences

This option specifies which character(s) will identify the separator(s) between a Sentence End and a Sentence Begin. Only characters from Classes S and N are allowed, and the number of characters required signifies the separation, e.g. 2SN means 2 characters from Class S or one from Class N. This option must contain at least one character from either class, but cannot exceed nine characters for each of the classes. The default system values are SN (or 1S1N), meaning that in order for two legal sentences to be legally separated, there can be either one space or one character from Class N between them.

### Characters classes that begin sentences

This option specifies which character(s) will be used to identify the beginning of a sentence, and can include one or more values from the character Classes L, U, D, H or B. Class B can be used to specify characters that are not present in the predefined classes L, U, D or H. This option must include between one and five classes. The default system classes for this option are UB.

### Defining a Paragraph

The fields shown below allow you to define the beginning, the end and the separations between sentences.

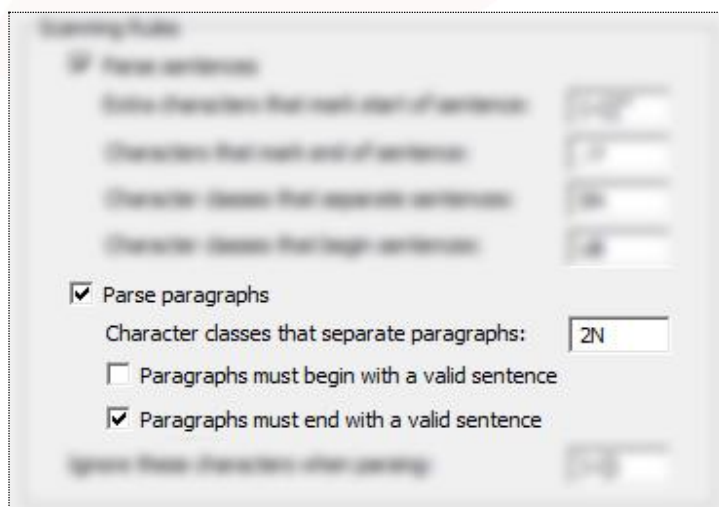


Figure 2 - 4 The Paragraph Parsing fields

The default values for paragraph parsing are shown in Figure 2 -19 above. A more detailed description of these conditions follows.

### Parse paragraphs checkbox

Checking or clearing the 'Parse sentences' checkbox will respectively switch paragraph parsing on and off

Paragraph separation is recommended for performance reasons, since a single-character modification in a document that exists as a single large block of text would require reindexing of the entire document. If paragraph separation is operative, only the altered paragraph in the document needs to be indexed again.

As with Sentence Separation, if incoming data quality is uncertain you may wish to deactivate Sentence Begin/Sentence End Required and use the Paragraph Separation classes to section the data. If you have filled your database without specifying Paragraph Separation and change this option to Yes, you will need to rebuild the database as discussed previously.

### Character classes that separate paragraphs

This option specifies which character(s) satisfy the requirements for a paragraph break, the default being two characters from Class N, or 2N (usually 2 <LF>). The number of characters required from Class N followed by the number of characters from Class S defines a Paragraph Separation, e.g. 2N4S means two characters from Class N followed by four characters from Class S. The number of characters from Class N must be greater than the number of characters from Class S for Sentence Separation (if any has been specified). This option must contain at least one character from Class N, but cannot exceed nine characters for each of the classes.

To continue with the example from Sentence End usage:

```
[A] ..... Mother Goose Rhymes:      <CR><LF>
..... <CR><LF>
..... (1) The cow jumped over the moon.  <CR><LF>
..... <CR><LF>
..... (2) Old Mother Hubbard lived in a cupboard.
..... <CR><LF>
..... <CR><LF>
..... (3) Hickory Dickory Dock, the mouse ran up the
clock. .... <CR><LF>
```

If the Paragraph Separator is 3N (three new lines), then [A], (1), (2) and (3) constitute a single paragraph; if 2N, then each of them forms one new paragraph.

### Paragraphs must begin with a valid sentence

This option specifies whether a sentence begin character is required, i.e. whether a character from the Sentence Begin classes (as specified above Sentence Separation) must be received to complete a paragraph break. Alternatives include Y (Yes) or N (No); the system default is N.

Again using the preceding example:

```
[A] ..... Mother Goose Rhymes:      <CR><LF>
..... <CR><LF>
```

```

..... (1) The cow jumped over the moon.    <CR><LF>
..... <CR><LF>
..... (2) Old Mother Hubbard lived in a cupboard.
..... <CR><LF>
..... <CR><LF>
..... (3) Hickory Dickory Dock, the mouse ran up the
clock..... <CR><LF>

```

If Sentence Begin Required is No (where a paragraph need not begin with a valid sentence begin character), the Paragraph Separator Class is 2N, and Class B is not included in the Sentence Begin classes, then [A], (1), (2) and (3) represent individual paragraphs. If the Paragraph Separator class is 3N, then they form a single paragraph.

If Sentence Begin is Yes (where a paragraph must begin with a valid sentence begin character) and Class B is not one of the Sentence Begin classes (which it is by default), then [A], (1), (2) and (3) form a single-sentence paragraph.

Therefore, a paragraph separation can be defined as:

- 1 a text block
- 2 one character from Class E (if Sentence End is required)
- 3 specified number of characters from Class N
- 4 the specified number of characters from Class S
- 5 one character from the Sentence Begin classes (if Sentence Begin is required)
- 6 a text block, as illustrated in the table that follows:

Text Block	1 from Class E	? from Class N	? from Class S	1 from Begin	Next Block
ABC0123 . .	. ! ?	<LF><VT><FF>	ASCII 0-32	ABC(<["	DEF4567 . .

**Table 2–5 Paragraph definition in TRIP**

The illustration above shows two paragraphs and their separators.

### Paragraphs must end with a valid sentence

The options are checked for Yes and cleared for No; the system default is Yes.

This option specifies whether a Sentence End Character is required (i.e. a character from Class E must be detected) to indicate the beginning of a new paragraph, or paragraph break.

The following outline illustrates Sentence End usage:

[A] ..... Mother Goose Rhymes: <CR><LF>  
.....<CR><LF>  
..... (1) The cow jumped over the moon. <CR><LF>  
.....<CR><LF>  
..... (2) Old Mother Hubbard lived in a cupboard.  
.....<CR><LF>  
.....<CR><LF>  
..... (3) Hickory Dickory Dock, the mouse ran up the  
clock.....<CR><LF>

The dotted lines in the illustration above represent white space.

If Sentence End is defined as Yes and the colon [:] is not part of the Sentence End classes, then Sentence [A] and Sentence (1) constitute the same paragraph. If Sentence End is No and the [:] is not included in the Sentence End classes, then Sentences [A] and (1) exist as separate paragraphs, depending on the Paragraph Disconnection and Sentence Begin Required options that were chosen.

*Note:*

*Exercise caution when choosing characters designating a Sentence End. Adding characters that occur frequently in normal text (such as colons, parentheses etc.) may cause problems in text division.*



### Setting characters to ignore

The fields shown below allow you to define the beginning, the end and the separations between sentences.



**Figure 2 - 5 The Ignore Character field**

The default values for ignoring are shown in Figure 2 -19 above. A more detailed description follows.

### Ignore these characters when parsing

This option specifies which character(s) will belong to Class I (Ignore characters). If characters are removed from Class I, and not placed in Classes B or E, they will be moved to Class R. This option cannot exceed sixty-four characters. The default system values are ) > ] } »

Therefore, a sentence separation can be defined as:

- 1 a text string
- 2 one character from Class E
- 3 default sentence separation (characters from Classes S or N) or user-defined separation
- 4 one character from the Sentence Begin classes (one or more of L, U, D, H or B)
- 5 a text string, as shown below:

Text String	1 from Class E	1 from Class S/N	1 from Begin	Next string
ABC0123 . . .	. ! ?	␣ <LF>	ABC(<["	DEF4567 . . .

**Table 2–6 Sentence definition in TRIP**

The illustration above shows two sentences and their possible separators, where ␣ represents a single space.



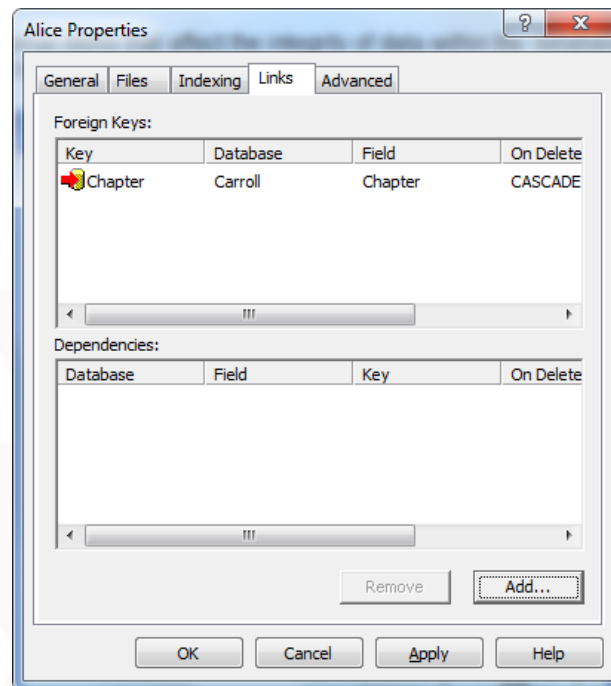
### Considerations for Altering Scanning Rules

- Be careful when changing the default settings for a pre-existing database. You must extract all data before the changes are made and reload it into the new design.
- If Sentence Separation is required without Paragraph Separation, then TRIP will automatically set the Sentence End required option to Yes, Sentence Begin required option to No and clear the Paragraph Separator classes option.
- If Paragraph Separation and Sentence Begin and/or End is required without Sentence Separation, then the Sentence End and Begin classes options must still be specified.
- Characters of Class I are ignored if found after the character that initiates a sentence or paragraph break (normally a Class E character), but before the break is complete.
- Characters from Class R, if found in the same position as above, will inhibit the current break and cause the program to scan for the next character that will commence a sentence or paragraph break.
- Characters from Classes L, U and D that are not specified as Sentence Begin classes will be treated as belonging to Class R.
- Characters from Class H that are not specified as Sentence Begin classes will be treated as belonging to Class I.
- Characters from Classes S and N will be treated as belonging to Class I when they do not appear as Sentence or Paragraph Separators.

## Database Properties (4) – Links

The link properties for a database comprise items that affect the integrity of data within the database, and how that integrity should be maintained and/or enforced with regard to other databases in a multi-database system.

Clicking on the 'Links' tab of the Properties will change the display to show the Database Links Property form. For example:



**Figure 2- 6 The Database Links Properties Form**

In this case, the DBA established a link between the ALICE database and the CARROLL database, stating that the "Chapter" field in both databases holds a shared value (a "foreign key" in relational database terms). Further, the link specifies how that sharing is to be enforced in the case of updates or deletes -- in this case, deletes are not allowed, whilst updates are to be cascaded through the link.

Links in general exist to stop data in one database getting out of synch with data in another database. For example, imagine that you setup a database holding code/name pairs:

CODE	NAME
ABC	ACME Broadcasting Corporation, Inc.
MOX	Ministry of Xenological Affairs
RAZ	Rationing Association of Zodiacs, Corp.

Now further imagine a production database that stores the code within its own data and uses this "lookup" database when reporting to translate codes into names. All's well until at some later point, you decide to update the code "MOX" to "MOXA".

Users confronted with the above situation have two options when searching (through a MAP, for example) for the Ministry:

- Always add wildcard prefix and suffix notation to every search they ever do, just in case the code changes slightly
- Report the problem to the DBA, who has to resort to a global update to correct the issue

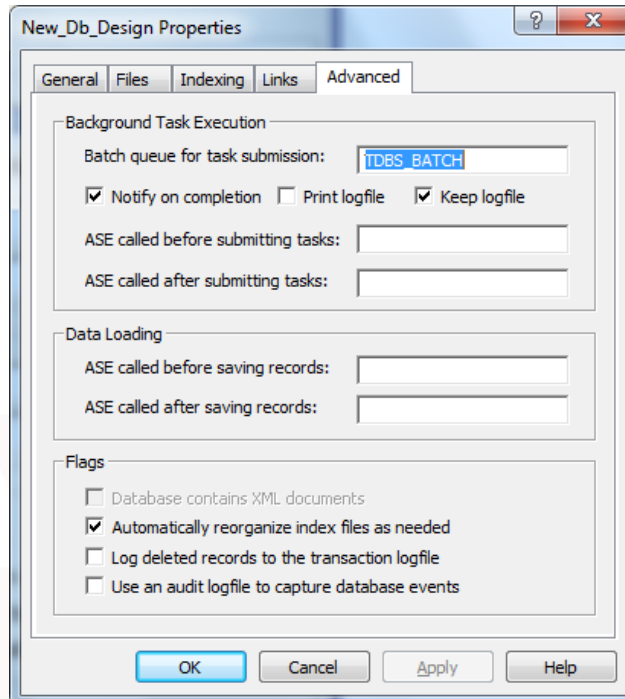
Using a link allows the DBA to specifically allow or deny certain operations, and to ensure that if updates or deletes are performed on the "lookup" database (in this case), those updates or deletes are reflected back onto the production database.

The types of action that you can specify to take place in the face of either an update or delete are:

- **RESTRICT** -- disallow the operation if it would affect any records in the linked database
- **CASCADE** -- reflect the update (or delete) on the linked database -- for example in our case described above, if the code "MOX" were updated to "MOXA", then when the record is committed to the "lookup" database, the production database is also updated automatically to change every occurrence of "MOX" to "MOXA".
- **SET\_NULL** -- any records containing the affected value will be blanked in the linked database.
- **SET\_DEFAULT** -- any records containing the affected value will have that field reset to its default value, whatever that might be.
- **NO\_ACTION** -- the default (and the traditional behaviour of TRIP).

## Database Properties (5) – Advanced

Clicking on the 'Files' tab of the Properties will change the display to show the Database Files Property form:



**Figure 2- 7 The Database Advanced Properties Form**

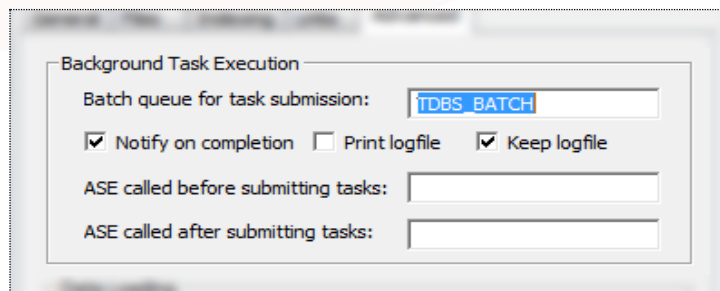
The final tab is the 'Advanced' properties tab. This tab has three areas:

- Background Task Execution
- Data Loading
- Flags

These areas are describe in detail in the following sections.

### Background Task Execution

This feature allows different indexing and queue submission criteria to be defined for each database, and is used when loading, global updating and indexing jobs are submitted from TRIP.



**Figure 2- 8 The Background Task Execution form**

### Batch queue for task submission

This is a logical name by which the TRIPdaemon separates and serialises execution of tasks on different databases. The default for all platforms is "TDBS\_BATCH"; this is simply a "catch-all" queue.

### Notify On Completion

If this flag is set, the TRIPkernel will signal to the TRIPdaemon that when the background task completes, it should notify the end user of this fact. Depending on the platform being used, this notification is more or less successfully delivered. Typically in a web-based environment, for example, such notifications are lost.

### Print Log File

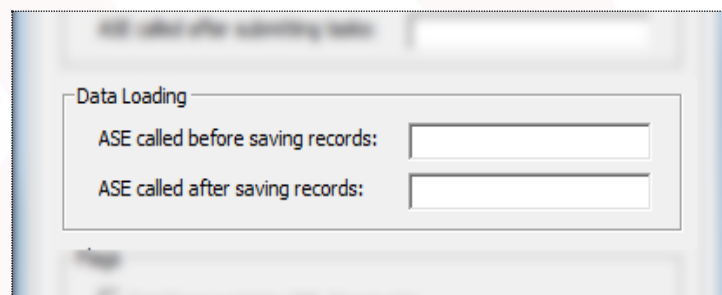
If this flag is set, any log file generated by a background task will automatically be submitted for printing when the task completes.

### Keep Log File

If this flag is not set, any log file generated by a background task will automatically be deleted when the task completes.

### Data Loading

This section of the Database design properties 'Advanced' form defines the calling of any ASEs necessary for loading of TForm records.



**Figure 2- 9 The Data Loading form**

Here you may enter the names of any ASEs you wish to call either before or after saving records to the BAF.

### ASE To Be Called Before Submission

An ASE can be called after all processing for a TRIP job has completed and before it is submitted to the batch queue, allowing customized checks to be incorporated into the submission process. Consult the Appendix in this manual for further information.

### ASE To Be Called After Submission

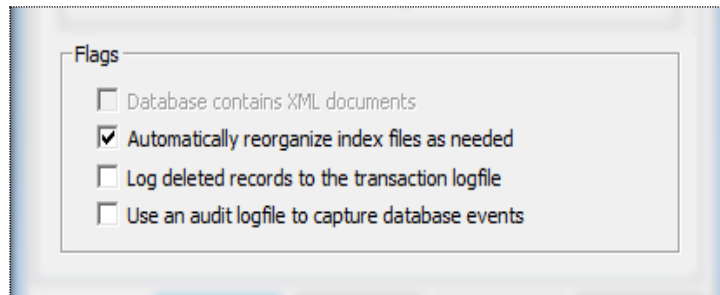
An ASE can be called if and after the job has been submitted to the batch queue specified. For further details, refer to the Appendix in this guide.

*Note:*

*See Appendix C of this manual for more information regarding ASEs.*

### Flags

The final section of the Database design properties 'Advanced' form allows for the setting of four flags.



**Figure 2- 10 The Database Flags Form**

The flags are as follows:

#### **Database contains XML documents**

If set, TRIP allows path-specific searches to be accomplished that take advantage of the structure of the XML documents stored within the database. Without this flag set, any XML documents stored in the database are simply treated as text.

*Note:*

*XML can only be performed at the moment of database creation; hence this checkbox is automatically greyed out and cannot be altered. If the database was created as XML enabled, then the checkbox will be checked. If not, the checkbox will remain unchecked.*

#### **Automatically reorganise index files as needed**

It is not uncommon in large database environments for an update to the database to require significant physical storage and time as that update causes a reorganization of the index files. This process can significantly impact both the performance and the availability of the database, and so administrators can clear this flag to stop the TRIPkernel from performing such automatic reorganization. When the database does need reorganization, the indexing log files will reflect the need and the administrator must then run the TRIPkernel utility REBIF to perform the reorganization (for detail on the usage of this utility, consult REBIF.PDF in the TRIPsystem installation).

*Note:*

*This checkbox is set by default.*

#### **Log deleted records to the transaction log file**

Set this flag to force the content of deleted records to be logged in the transaction log file. This can be useful to administrators when attempting to decide if a given record should be reinstated to the database, or if a user inadvertently deletes a record and then wishes to recover it. In such a circumstance, the administrator can simply locate the record in the log file, extract the contents and minimally edit them to turn it from a delete to an update.

#### **Use an audit log file to capture database events**

In certain extreme circumstances, notably when tracking down complex bug scenarios, it might be necessary to be able to see every search performed against the database. Establishing an audit trail is the means by which this is made possible.

Simply checking this option does not turn on the audit file and this option is not supported unless you are being guided by customer support, as there are

other steps that must be undertaken as well. This policy is for the protection of database administrators, as unwittingly turning on this option will significantly degrade the performance of any operations on the database.

## Field Definition

### Defaults and Restrictions

The following table presents the defaults and restrictions (and defaults if defined) for each of the TRIP field types TExt, PHrase, NUmber, INteger, DAta, TIme and STring.

Restrictions	TE	PH	NU	IN	DA	TI	ST
Field name	A	A	A	A	A	A	A
Field type	A	A	A	A	A	A	A
Field is included in index	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X
Layout retained	<input checked="" type="checkbox"/>	<input type="checkbox"/>	X	X	X	X	X
Part field	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Field can have subfields	A	A	A	A	A	A	A
Can have valid value type	X	A	X	X	X	X	X
Can have valid value	X	A	A	A	A	A	X
Description	A	A	A	A	A	A	A

**Table 2–7 Field Defaults and Restrictions**

- ☒ Default: Checkbox is checked
- ☐ Default: Checkbox is cleared
- A Applicable for this data type
- X Not applicable for this data type

### The Modify Fields Collection Form

If you chose yes to the dialogue box presented at the end of the database design wizard, you will be immediately taken to the 'Modify Fields Collection' form. If you chose 'No' in response to the dialogue box, then you will need to select the database for which you wish to create fields and choose the 'Modify Fields Collection...' action from the menu.

*Note:*

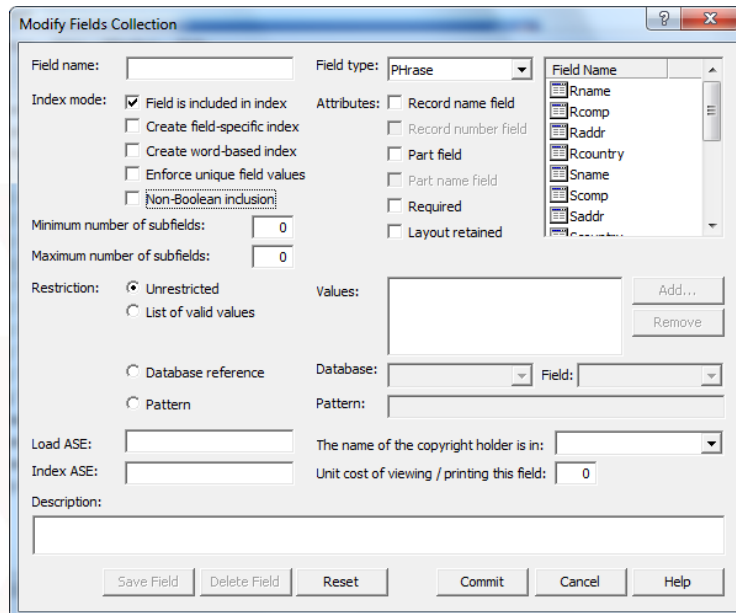
*Depending on which field type is selected and which options for that field are chosen, different areas of the 'Modify Filed Collections' form will automatically be made available or unavailable as appropriate.*

For example:

- The 'Record name field' selection box will only be unavailable if the field is of type, Phrase.

- The 'Record number field' selection box will only be unavailable if the field is of type, Integer.
- If the selection box for 'Field is included in index' is left cleared, then the other two selection boxes in the 'Index mode' area of the form will be unavailable.

Once you chose to modify the fields collection, you will be presented with a form similar to figure 2-10 overleaf, showing the default state of this dialog with a simple database design loaded (in this case, the example database CORR):



**Figure 2- 11 Modify Fields Collection Form**

There are two basic mechanisms for interacting with the fields collection through this dialog:

### **Editing or deleting existing fields**

In order to modify or delete an existing field, select it in the list to the top right of the dialog. Selecting such a field will cause the field's properties to be displayed in the dialog.

To edit the field's properties, simply make the changes required and click the "Save Field" button.

*Note:*

*The "Save Field" button will not be available until a valid modification is made).*

To delete the field, simply click the "Delete Field" button

*Note:*

*The "Delete Field" button will only be available if the database is empty.*

In order to clear the dialog back to its default state (for example, after loading an existing field and before creating a new one), click the "Reset" button.



When you have completed your changes to the fields collection, you must commit the changes to the database design. To do this, simply click the "Commit" button.

### Creating new fields

In order to create a new field, simply specify the field name and choose the type from the drop-down. Everything else is optional and will default to "sensible" values. If you have no need to customise the field further, simply click the "Save Field" button and the new field will be added to the list of fields shown in the control at the top right of the dialog.

The controls on the dialog are described in the following sections, starting with the field name entry box and the field type drop-down selection box:

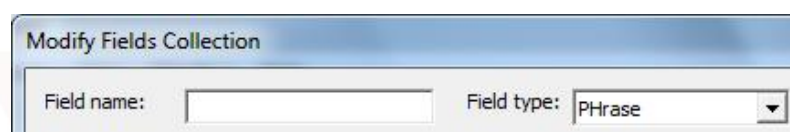


Figure 2- 12 Field Name Entry and Field Type Selection

### The Field List

After defining or modifying (and then saving) one or more fields, they will be visible in the field list window on the 'Modify Fields Collection' form.

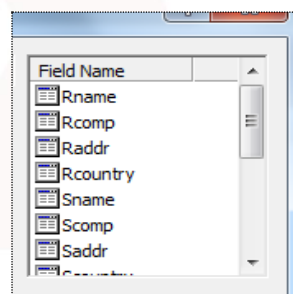


Figure 2- 13 The Field List

To select a field from the list for closer inspection or alteration, simply click on it. To save any changes, click on the 'Save field' button. To leave the list without selecting, click on the 'Reset' button; this will reset the form to its default state.

### Field Name

Choose a field name which is unique to the parent database for the new field you wish to define, and type it in the 'Fieldname' box.

*Note:*

*A field name in TRIP may contain from one to sixteen alphanumeric characters, must start with a letter and may include the underscore ( \_ ).*

### Field Type

Use the drop down selection box to select one of the following field types; either, TExt, PHrase, NUMber, INteger, DAte, TIme or STring.

Depending on the field type chosen, the rest of the dialog will be initialized accordingly, so as to only make available those options that are valid for the field type.

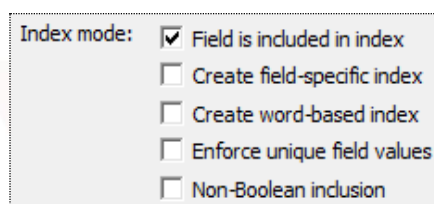
*Note:*

*When editing fields, you cannot modify the type of a field if the database is not empty.*

### Index Mode

Indexing makes the data in each field available for searching. If a field is defined as 'Not Indexed', the contents can only be displayed, not searched.

The indexing modes are Indexed, Unindexed, Field-specific Indexing and Word-based Indexing.



**Figure 2- 14 Index Mode Selection**

You can set the indexing mode by checking, or unchecking, one of three checkboxes described below:

#### Field is included in index

Checking this checkbox sets the indexing category to Indexed. Leaving the checkbox unchecked sets the category to Unindexed.

Indexed is the default or normal indexing mode, in which each word, phrase and their component grams (uni-, bi- and trigrams) are indexed.

For TExt fields, each word and word gram is indexed separately.

With PHrase fields, each complete phrase is indexed as an individual term in addition to its words and word grams. This allows use of the CCL construct

```
Find fieldname='a phrase' ↵
```

which performs exact matching for 'a phrase' on an entire subfield of a PHrase field.

The numeric field types (INteger, NUMber, DAte and TIme) are indexed by field number rather than field content.

SString fields are non-indexable; however, the field number is indexed to allow for use of the CCL construct

```
Find fieldname=$ ↵
```

which will find all records where fieldname has content.

#### Create field-specific index

This checkbox selection is available only for PHrase and TExt fields. Terms occurring in a field so designated will have a default index plus an additional posting list (separate index) for occurrences in this field.

This is valuable in those instances where the field contents contain identically-spelled homonyms of terms commonly found in the general

database. For example, many applications use code words or acronyms that are also common words in the general vocabulary, but have a different meaning, such as the stock exchange company identifiers 'THE' and 'IBM' (Figure 2-28). Since these business abbreviations may occur far more frequently in the TExt fields of the database at large than in this particular field, separate indexing would save searching time and resources.

Term	Default List							Separate Index							
	Number of Records	Number of Occurrences	Postings: Individual Occurrences					Number of Records	Number of Occurrences	Postings: Individual Occurrences					
			1	2	3	...	n			1	2	3	...	n	
STD	8903	9706	---	---	---	---	---	103	189	---	---	---	---	---	
THE	115852	273451	---	---	---	---	---	52	67	---	---	---	---	---	
IBM	4371	5724	---	---	---	---	---	371	424	---	---	---	---	---	
SAT	1289	2833	---	---	---	---	---	12	33	---	---	---	---	---	
DIN	<div>Record number</div> <div>Field number</div> <div>Subfield or paragraph number</div> <div>Sentence number (TExt fields only)</div> <div>Word number</div> <div>Word position</div>		---	---	---	---	---	23	27	---	---	---	---	---	

**Figure 2- 15 Separate Indexing**

Each posting box contains a record number, field number, subfield or paragraph number, sentence number (in a TExt field), word number and word position, all of which is represented by dashes in the illustration above.

### Create word-based index

This option applies to PHrase fields only, where words and their corresponding grams are indexed rather than the entire phrase. This is advantageous in those instances where you wish to maintain the utility of a PHrase field, but have no need to Display entire phrases. Word indexed PHrase fields are displayed in the same manner as TExt fields, i.e. words are shown individually rather than as constituents of phrases.

This category saves file space; however, exact matching during searching is not possible.

### Enforce Unique Field Values

If the field is of type PHrase, you can select this checkbox to make TRIP check each entry into this field to ensure that it occurs in no other record in the database.

### Non-Boolean Inclusion

Check this flag to include the contents of this field in all Non-Boolean calculations, including processing required for the ABOUT() search function and also for document classification.

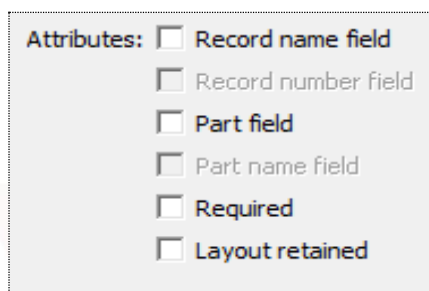
Notes:

- For more detail on using the ABOUT()function, consult the "Define" and "Find" sections in the "CCL Command Reference", included with the TRIPsystem documentation.
- For more detail on non-Boolean searching, see the white paper entitled, "TRIP Non-Boolean Searching", included with the TRIPsystem documentation.

- For more detail on how to create and manage classification schemes, see the relevant section in Appendix B and also the white paper entitled, "TRIP Document Classification", included with the TRIPsystem documentation.

### Field Attributes

The Field Attributes include whether the field is a Record name (or number) field, is a Part Field, is Required or is Layout Retained and are available depending on which field type is selected. E.g. The figure below shows the default attributes for a Phrase type field.



Attributes: ☒ Record name field  
☐ Record number field  
☒ Part field  
☐ Part name field  
☒ Required  
☒ Layout retained

Figure 2- 16 Field attributes selection

### Record name field

TRIP automatically numbers each record with a unique record identifier as it is entered into the database. In some applications it may be desirable to have a record name field as well. This must be a PHrase field of 255 characters or less, and its contents in a record must be unique in the database.

For example, a technical reports database system may have a record name field consisting of three parts: a location code, a database name code and a document number. This three-code arrangement will produce a unique record name value for each technical report in the system, as seen below:

Location	Database	Report Number	Record Name
Chemistry	Organic_Chemistry	0001984	Chemistry_Organic_Chemistry_0001984
Chemistry	Physical_Chemistry	0023132	Chemistry_Physical_Chemistry_0023132
Chemistry	Nuclear_Chemistry	0005767	Chemistry_Nuclear_Chemistry_0005767

Table 2–8 Use of the record name field

Although this feature can be extremely useful, it is widely overused. Nonjudicious inclusion of record name fields in a database design results in needless performance degradation during data loading, as both the BAF and the BIF must be updated.

### Record number field

The record number given by the system may be put in a record number field. The contents of that field will be searchable in the usual way, but cannot be changed.

Record number fields are not necessary with new database designs. They are included here solely to maintain compatibility with older designs, for which the CCL construct

```
find R=n ↵
```

was not available.

### Part

Checking this checkbox makes this field a part field in a part record; Leaving it unchecked makes it a head field in a head record.

The default for this design field is unchecked and if no fields in the database have been designated as part fields, this becomes a 'head' or 'flat' record. For more information on part fields, see Chapter 2, 'Records'.

*Note:*

*When editing fields, this value can only be changed in an empty database.*

### Record Part Name Field

If your database design employs head/part record structure, you should always utilise record part name fields, which allow the unique identification of parts within a composite record.

### Required

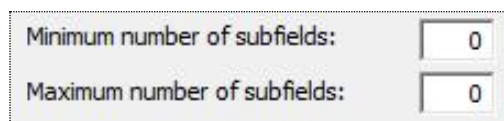
This is a courtesy flag that simply sets the minimum number of subfields or paragraphs to 1 (or to zero, if the checkbox is cleared). This checkbox can be modified at any time, although doing so has no effect on records already in the database.

### Layout Retained

For TExt fields, Layout Retained ensures the preservation of formatting characteristics such as <Tab>, <LF>, and blank lines in the BAF, exactly as they occur in the entered text. In PHrase fields, <Tab> and multiple spaces are maintained.

The default for PHrase fields is No, meaning that a series of spaces or tabs will be compressed to a single space when the record is loaded into the BAF. The default for TExt fields is Yes. A No in a TExt field causes <Tab> to be converted into a single space, multiple spaces into a single space and carriage returns/line feeds to be removed.

### Field Organisation (Subfields and Paragraphs)



Minimum number of subfields:	0
Maximum number of subfields:	0

**Figure 2- 17 Subfield specification**

The above two fields accept up to three digits each, and can be used to define the field organisation of any data type except String – See note below.

They allow you to define the minimum and maximum number of subfields allowed for all field types except TExt, for which they specify the minimum and maximum number of paragraphs.

Assigning a TExt field a maximum means that it must not contain more paragraphs than that number. If a maximum of one paragraph is defined and that definition is changed after data has been loaded into the database, the database must be rebuilt.

If the minimum number of paragraphs is one or more, the field must contain at least that number of subfields, and entry into that field is mandatory.

*Note:*

*Due to their internal structure, it is strongly recommended that STring type fields not be limited to a set number of subfields or paragraphs, as to do so will result in an error message when trying to write paragraphs of greater than 1MB in size.*

### Setting Field Restrictions

All field types are unrestricted by default. However, if required, field restrictions can be set by selecting one of the three radio buttons; 'List if valid values', 'database reference' or 'Pattern'.

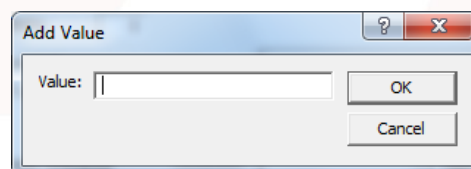
#### Valid Values

You may specify a comma separated list of accepted values for a field by clicking on the 'List of values' radio button and then clicking on the 'Add' button.



**Figure 2- 18 List of valid values**

This will cause an entry box to appear:



**Figure 2- 19 Valid values entry box**

Type the desired values into the entry box and click on the 'OK' button to add them to the list of values in the 'Values' text area.

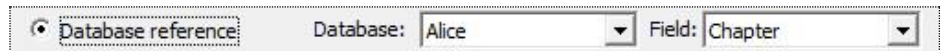
Each element in the list represents the contents of a subfield. The list may contain at most twenty elements, for a maximum length of 255 characters.

To delete an unwanted entry from the list, simply select it from the 'Values' window and click on the 'Remove' button.

### Database Reference (Dictionaries)

Selecting the 'Database reference' radio button signifies that the field currently being defined is to be restricted by a database reference ('Data dictionary') that is held in a specific field of a particular database.

For example, selecting the field 'Chapter' in the database 'Alice' using the database reference drop-down selection boxes:



**Figure 2- 20 Database reference selection**

means that entries are validated against the field 'Chapter' in the database 'Alice'.

*Note:*

*To specify that entries must NOT occur in the dictionary, use the NOT modifier, e.g.*

*NOT Alice.person*

*in the Valid Value field (Figures 2-18 and 2-19).*

*The dictionary can also consist of fields from more than one database, for example:*

*Alice.person, Alice.speaker, Corr.rname*

*where the fields are combined with OR operators.*

*In this way an entry found in only one of the fields is enough to permit entry of a term into the database.*

### Pattern

A pattern is a regular expression giving the total number of words or characters and the set of characters themselves that may be entered in each subfield of the PHrase field.

Patterns represent the fastest method of data validation available in TRIP, and are especially useful with figures such as inventory control or part number data. For example, a pattern for the TRIP version number V2.4–11 could be defined in this way:

Letter V + digit + . (period) + digit + optional - (hyphen) + optional digit(s)

Patterns may be disadvantageous to the database designer in those instances where the complexity of the pattern itself dictates pattern development costs that far outweigh any potential benefit. They may also be confusing to the user who, when attempting to access data entry Help for a pattern-controlled field, will receive only a display of the pattern the field is restricted against, without explanation.



**Figure 2- 21 Pattern entry**

A pattern is defined by selecting the Pattern radio button and entering the valid pattern in the entry field next to it. There can be only one pattern for a PHrase field.

A pattern may consist of a number of parts, including the following:



Symbol	Usage
( )	control characters indicating start/end of word pattern
*	start of character set specification
w,e,s,x,o, 9	letters are case-insensitive (see table below)
..	'from . . to'
/	named characters following / are to be added to or deleted from set
//	characters following // are identified by ASCII values
+	add characters to character set
-	subtract characters from character set

**Table 2–9 Symbols used in pattern specification**

An asterisk [\*] marks the start of a character set specification. A word pattern consisting of a single word pattern part with no character set specification (i.e. without the asterisk) specifies a list of words. For example, (2..\*9) represents a word pattern, while (2) designates two words.

Each word pattern part must be surrounded by parentheses, if it contains more than one or more explicitly given characters that are to be matched. This is a generic clause, and indicates that more than one possible character will be accepted for any one position in a pattern. All multinational characters are valid as matching characters as long as the string does not conflict with what is stated within parentheses. Letters are not case sensitive, neither as matching characters nor in a character set specification (an a in the pattern will accept both A and a). Any character will match itself.

An interval is symbolized by two dots [..], and character sets are represented by letters and digits (see table following). A slash [/] immediately following a character set name signifies that explicitly given characters should be added to [+] or subtracted from [-] the given character set. A double slash [//] directly after a character set name signifies that the characters following it will be given by their ordinal numbers in the DEC multinational character set.

The six predefined character sets are outlined in the table below.



Symbol	Character Set
w	digits and multinational letters
e	English letters
s	Swedish letters
x	non-space characters
0	empty set
9	digits

**Table 2–10 TRIP’s predefined character sets**

The simplest patterns are those which require one or more characters from any character set to be entered, for example:

Expression	Character	Function
(1*e)	(	start word pattern part
	1	exactly one character required
	*	‘of’ or ‘from’
	e	the English letter character set
	)	end word pattern part

**Table 2–11 A simple pattern**

For example, to ensure that a data value will contain three English letters followed by a slash, then three digits, a hyphen and a single digit (for example, abc/123-7), the pattern (3\*e)/(3\*9)-(1\*9) would be entered without spaces or commas:

Expression	Character	Function
(3*e)	(	start word pattern part
	3	three characters required
	*	'of' or 'from'
	e	the English letter character set
	)	end word pattern part
/		slash required at this position
(3*9)	(	start word pattern part
	3	three characters required
	*	'of' or 'from'
	9	the Digits character set
	)	end word pattern part
-		hyphen required at this position
(1*9)	(	start word pattern part
	1	single character required
	*	'of' or 'from'
	9	the Digits character set
	)	end word pattern part

**Table 2–12 A more complex pattern**

Examples of some easily applied patterns follow.

Pattern	Coding	Meaning
12	(2..*9)	Strings of two or more digits.
a4c/123-7 or 1b3/123-7 or abc/123-7 or 123/123-7	(3*e+9)/(3*9)-(1*9)	Three English letters, a slash, three numeric characters, a hyphen and one numeric character.
abc/123-7 or ab/123-7 or a/123-7	(1..3*e)/(3*9)-(1*9)	One to three English letters, a slash, three numeric characters, a hyphen and one numeric character.
a...z/123-7	(1..*e)/(3*9)-(1*9)	One to 249 English letters, slash, etc.
/123-7	(0..*e)/(3*9)-(1*9)	Any number of English letters, slash, etc.
abc or a12 or 12a	(3*e+9/-09876543) or (3*e+0/+12)	Three English letters or digits or a 1 or 2, slash, etc.
a(b)	(4*e+0//+40,41)	Four English characters or parentheses.
a b c	(1..3)	Strings of one to three words.
a1!b2@c3#d	(10*x)	A word of ten characters.
äåö	(3*s-e)	Three letters from the Swedish character set minus the English set.
äåö äåö 012	(1..*s) (0..*s) (1..3*9)	Strings of one or two words of Swedish letters, and one word of one to three digits.
a ä æ	(1..*w) (0..*w) (0..*w)	Strings of one to three words of digits and/or multinational letters
A1b0c	a(1*9)(0..*e+9)	A string consisting of one word starting with the letter 'A' and a digit, optionally followed by a sequence of English letters and/or digits

**Table 2–13 More patterns**

Word patterns can specify sets of legal characters for each of their parts. You can also use combinations of predefined sets with explicitly given characters added to or taken from the combinations. If no character set is specified, the set consists only of multinational characters.

*Note:*

*Remember what was said at the start of this section about the asterisk marking the start of a character set specification, and about the parentheses that must surround each word pattern part!*

Examples of combined sets are:

Character Set Configuration	Meaning
9+s	Digits and Swedish letters.
s-e	The three extra vowels in Swedish.
e+9/-089	The English letters and the digits 1 to 7
0/+AEIOU	The vowels a, e, i, o, and u.
9//+40,41	The digits and left and right parentheses

**Table 2–14 Sample combined character sets**

In the last example the parentheses are specified by their ordinal numbers in the DEC multinational character set; note the double slash.

The following paragraphs detail the valid field restrictions available to each field type.

### **Text Fields**

The contents of a TExt field cannot be restricted to valid values. For a PHrase field, however, there are several ways of doing this.

### **PHrase Fields**

A PHrase field may be restricted to valid values in three ways; by entering a list of valid values, by entering a reference field, or by entering a pattern:

### **NUmber**

A NUmber field may be restricted to a specified list of values and/or intervals, with the elements separated by commas. The list may contain at most twenty elements or 255 characters, and is entered in the entry field at the bottom of the screen. An interval is symbolized by two dots [..], in the same way as in a pattern specification for a phrase field.

A list of valid NUmber values might look like this:

-2..5.5, 20..

meaning that numbers between or equal to -2 and 5.5, and equal to or greater than 20 will be accepted. As this example contains a real number, it could not be used for INteger.

### **INteger**

An INteger field may also be restricted to a value/interval list, with at most twenty elements or 255 characters separated by commas. The list is also entered in the entry field at the bottom of the screen, with intervals symbolized by two dots [..].

A list of valid INteger values might look like this:

-2..5, 20..

meaning that numbers between or equal to -2 and 5, and equal to or greater than 20 will be accepted.

Another INteger restriction could be

..5, 7..

which would make all possible integers (with the exception of 6) valid.

### DAte

DAte fields may also be restricted to a specified list of values and/or intervals, the elements separated by commas. The length of the list is restricted in the same way as for a number; however dates may be specified using their full format (YYYYMMDD), year and month (YYYYMM) or year only (YYYY). A DAte list could look like this:

19850625..19851031, 1986

meaning that any date in 1986 as well as dates from June 25, 1985 up to and including the whole of October 1985 will be accepted.

If this date form is employed (digits, year followed by month and day, and no separators), any acceptable private date form may be used in data entry.

Restrictions given in other date forms will accept only dates of the same form in data entry.

*Note:*

*Refer to the section entitled 'Date Form' in Chapter 10 of the TRIPclassic User Guide for a listing of available date formats.*

### Time

The rules for Time fields are similar to those for DAte fields. A list of Time values and intervals could look like this:

10:10:00..13:30:10, 15

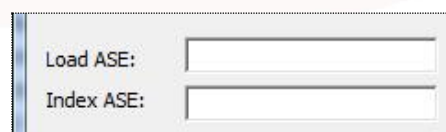
*Note:*

*The '15' in the example above indicates 3:00PM exactly (15:00:00), not 3:00:01PM to 3:59:59PM inclusive.*

We now continue with our description of the remaining 'Modify Fields Collection' form elements.

### Defining Field ASEs

Field ASEs are specified in the two 'ASE' boxes on the 'Modify Fields Collection' form:



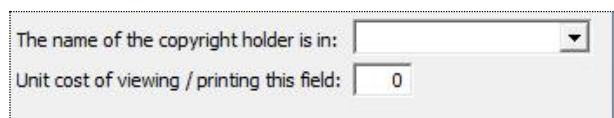
The image shows a screenshot of a software interface with two input fields. The first field is labeled 'Load ASE:' and the second field is labeled 'Index ASE:'. Both fields are empty and have a light blue border.

**Figure 2- 22 Field ASE specification boxes**

Here you can specify ASEs to be called on a per-field basis, either when loading a record from TForm or when indexing a record. An ASE is called once per field.

### Accounting Information

This part of the form allows the selection of a field containing the record copyright holder and a unit cost for examining the onscreen and printed contents of that field.



**Figure 2- 23 Accounting Information specification**

For example, if you download information from an online subscription service (host a copyrighted database), you could specify a copyright holder for the downloaded records for chargeback purposes, which will be reflected in the DEBIT.LOG file.

Refer to Chapter Four, 'System Logging Functions' for information regarding accounting functions, and the 'Output Format Reference Guide' section in Chapter Six of this manual for details on the <debit> filter.

### Description



This is a storage area for field descriptions and instructions regarding field content, and consists of free text up to 255 characters in length. Its contents will be displayed with the CCL commands **STatus**, **Show BAsE**, and **Print BAsE**, as well as during data entry if *field help* is activated.

## Saving a field design

To temporarily save the design for this field, before moving on to create or modify further fields, click on the <Save *field*> button, located to the bottom left-hand corner of the form. You will now be able to continue working with other fields in the database.

### Notes:

- *Saved new and modified fields will not be committed to the database until you either:*
  - a) *Click on the <Commit> button, to the bottom right of the form*
  - b) *Click on the <Cancel> button, to the bottom right of the form and respond <Yes> to the 'Save changes to DATABASE\_NAME design' confirmation dialogue.*
- *In both (a) and (b) above, the alteration of the database design will be confirmed by a pop-up TRIP message, 'Database design for DATABASE\_NAME altered'. Click the <OK> button, or press the <Enter> key on the keyboard to continue.*

## Committing field designs and changes to the database

To commit the designs for any new and modified fields to the database, click on the <Commit> button to the bottom left-hand corner of the form.

### Note:

*The alteration of the database design will be confirmed by a pop-up TRIP message, 'Database design for DATABASE\_NAME altered'. Click the <OK> button, or press the <Enter> key on the keyboard to continue.*

## Deleting a field design

To delete the design for a field, click on the <Delete field> button in the bottom left-hand corner of the form. You will now be able to continue designing the other fields for the database.

*Note:*

*If the field being deleted was previously committed to the database, a confirmation dialogue, 'Are you sure you want to delete FIELD\_NAME?' will first appear. Otherwise, if the field has been saved, but not yet committed to the database, it will be deleted without further warnings.*

## Saving a Database Design

Saving a new database design entails entering it as an item in the Control system file that controls the database. To save an entire design specification, press the <OK> button, or the keyboard <Enter> key from the General Database Properties form.

The BAF, BIF, VIF, and the Log file (if specified) are created when the database design is saved.

## Modifying a Database Design

You may alter anything in a database design as long as no data has been entered into the database.

If, however, data has already been loaded into your design, keep in mind the following recommended actions for implementing common design changes:

Type of Change Required	Recommended Action
Field name	None
File location	None
Default form (entry and output)	None
Searchable characters	Reindex database
Character folding	Reindex database
Index mode	Reindex database
Delete field from design	Rebuild database
Add/remove/change special field*	Rebuild database
Sentence/paragraph definition	Rebuild database
Field type	Rebuild database
Layout retained	Rebuild database
Field organisation	Rebuild database
Head/part status	Rebuild database

**Table 2–15 Modifying a database design**

\* The record name, record number and part name fields are special fields.

'Reindex database' means to reset the index status of the database by running \$TDBS\_EXE/bafini (%TDBS\_EXE%/bafini in Windows) and providing a database name at the prompt. You will then need to index the database.

'Rebuild database' means to print the database contents into TForm, rename the database BAF, BIF and VIF, alter the database design as necessary, load data into the modified database design from TForm, index the newly-filled database and delete the renamed BAF, BIF and VIF.

*Note:*

*If you change field names or types, you must edit all entry, output and search forms which refer to those fields to ensure uniformity among field name and data type references.*

To remove a field from the database design, bring up the 'Modify Fields Collection' form and choose the field to delete from the field list; then click on the 'Delete field' button; the deletion of a field is automatically committed to the database after Yes has been selected from the Yes/No prompt which appears.

*Note:*

*All currently open TRIP sessions will have to exit and re-enter before any changes that have been made become apparent; this is because, for performance reasons, TRIP caches the design of any open database for any given session and does not reread the design until a new session is started.*

## Deleting a Database Design

Select the database you wish to delete from the list of databases in the mmc window, and select 'Delete' from the Action menu.

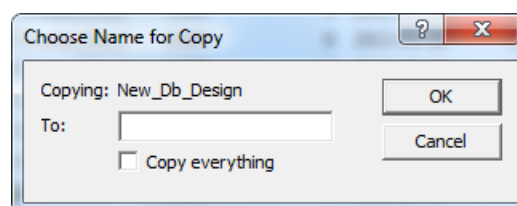
This will delete the database from the Control file as well as its associated BAF, BIF, or VIF files, as long as there is no data in the database.

If the database which has content, the delete option will not appear on the action menu, so you must first delete the database's BAF, BIF and VIF, after which all associated forms and formats will also be deleted.

## Copying a database Design

In the mmc main window, select the database you wish to copy and select 'Copy' from the 'Action' menu.

Next, select the 'Databases' root node in the TRIPmanager window, the select 'Paste' from the action menu. A dialogue box will appear, requesting the name for the new database copy:



**Figure 2- 24 Choose Name for Copy**



If you also wish to copy all forms and formats with the database design, check the 'Copy everything' checkbox. A confirmation dialogue box will then appear to confirm the copy.

*Note:*

*When attempting to perform the 'Paste part of the above operation, if you do not have the 'Databases' root node selected, the context sensitive 'Paste' option will not appear as it makes no sense to paste a copy of a database on top of an existing database.*

Once the new database copy has been created, you can alter it using the 'Modify Fields Collection' form.

*Note:*

*To create the files BAF, BIF, VIF and Log of the copy database, you must modify the database properties and save the design. This is important because, before the files are created, other users cannot be granted write-access to the database if the directory where the files are to reside is write-protected by the operating system.*

## Related CCL Commands

*Note:*

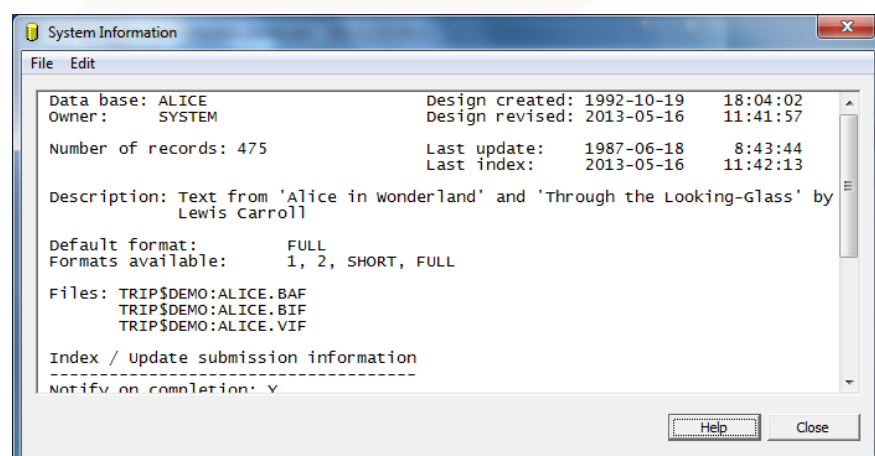
*It is necessary to select 'CCL Order', from the 'Action' menu, in order to be able to enter any CCL commands.*

## Status

When you have completed and saved the database design, you can look at the result by using the CCL command `Status` to review general information about your database.

The form of a `Status` order is

```
Status databasename ↵
```



**Figure 2- 25 Status for database Alice**

Here the BAF, BIF and VIF name specifications are given, as well as the names of the reports and those of any other forms belonging to the database. When there is data in the database, the total number of records and the dates of the latest database update and indexing are also provided.

The list of field names for that database will be presented in field number order, with their data types, field numbers (necessary for TForm data input), and comments, if any.

## Show

An overview of all databases created by a user is obtained by giving the CCL order:

```
Show BAsE
```

The list will contain, for each database, the same information as the SStatus command gives for a single database.

This order lists the databases in a shorter format than Show BAsE:

```
Show BAsE List
```

## Print

Use Print instead of Show to send output to a printer or file.

The system manager may add R=ALL to the order and have a list of all of the databases in the system, if the Control file has been indexed.

## IMPOrt and EXPOrt

To produce an ASCII definition file for a database design, use the CCL command EXPOrt:

```
EXPOrt BAsE=databasename [path=mypath]  
FIle=filename ↵
```

For example, on Windows:

```
EXPOrt BAsE=mybase path=C:\TRIP\exports  
FIle=myfile.def ↵
```

Or on UNIX:

```
EXPOrt BAsE=mybase.* path=/user/home/fred  
FIle=myfile.def ↵
```

to export the database design and any associated reports and entry forms for Mybase as well. If the (optional) path is not specified or does not exist, the file will be written to the local working directory.

This definition's description can be used to move the database description between Control files, or to create a new database description in the same Control file rather than using the Copy function. To use the definition file in this manner, use the CCL command

```
IMPOrt BAsE=databasename [path=mypath]  
FIle=filename ↵
```

For example, on Windows:

```
IMPOrt BAsE=mybase path=C:\TRIP\exports  
FIle=myfile.def ↵
```

or on UNIX:

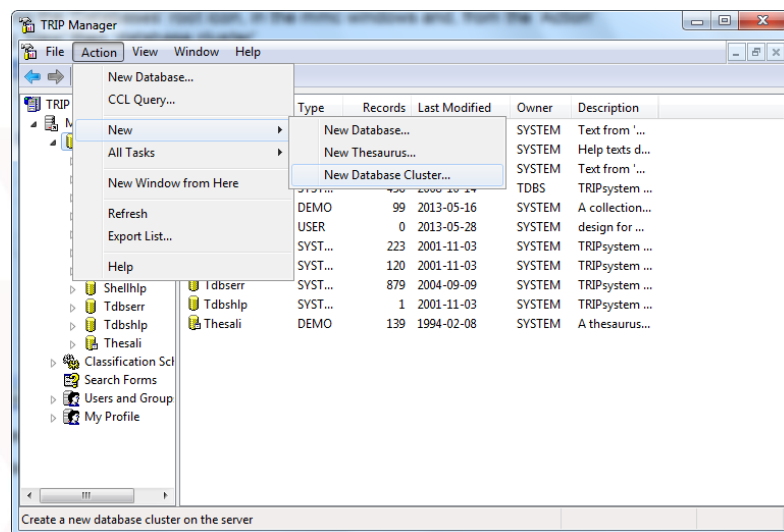
```
IMPOrt BAsE=mybase.* path=/user/home/fred  
FIle=myfile.def ↵
```

to import the database design and any associated reports and entry forms for Mybase as well. If the (optional) path is not specified or does not exist, an attempt will be made to read the file the local working directory. If the file cannot be located, a file not found error will be generated.

## Database Clusters

### Creating a Cluster

You can define static or permanent database clusters (as opposed to dynamically from CCL) by selecting the 'Databases' root icon, in the mmc windows and, from the 'Action' menu, selecting 'New' then 'database cluster'



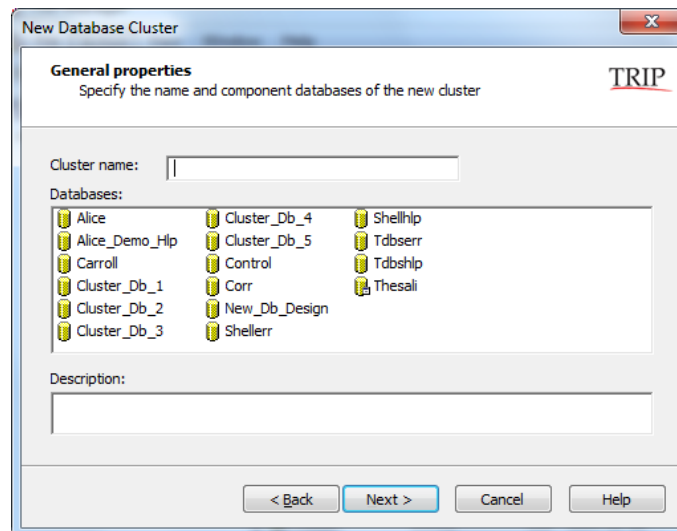
**Figure 2- 26 Create New Database Cluster**

The New Database Cluster' wizard will then start.



**Figure 2- 27 New Database Cluster Wizard**

The wizard will present you with a form in which to select up to thirty database cluster members.



**Figure 2- 28 Database selection form**

Enter a name for the database cluster into the 'Cluster name' entry box and select the desired databases (and / or other clusters) for the cluster, by holding down the <Ctrl> key on the keyboard, whilst clicking on databases in the 'Databases:' window.

Optionally enter a description in the 'Description:' window, to a maximum of 255 characters in length.

Finally, click on the 'Next' button to continue, or the 'Cancel' button to quite the wizard.

The finishing page of the wizard will appear. You can then click on the 'Finish' button to create the cluster.

*Note:*

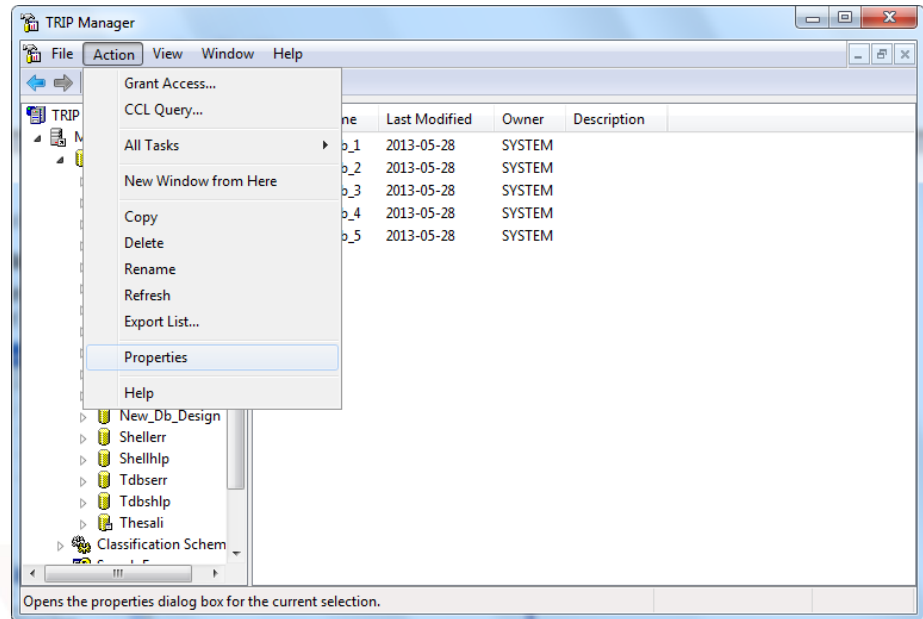
*The design of static database clusters is still limited to 30 members; however a workaround is to create database clusters containing other database clusters.*

*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.*

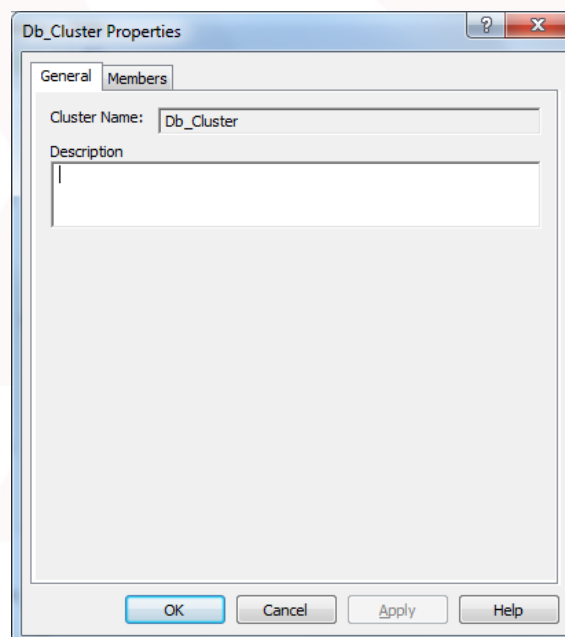
## Modifying a Database Cluster

In the mmc main window, select the cluster you wish to modify; then choose 'Properties' from the 'Action' menu.



**Figure 2- 29 Modify a Database Cluster**

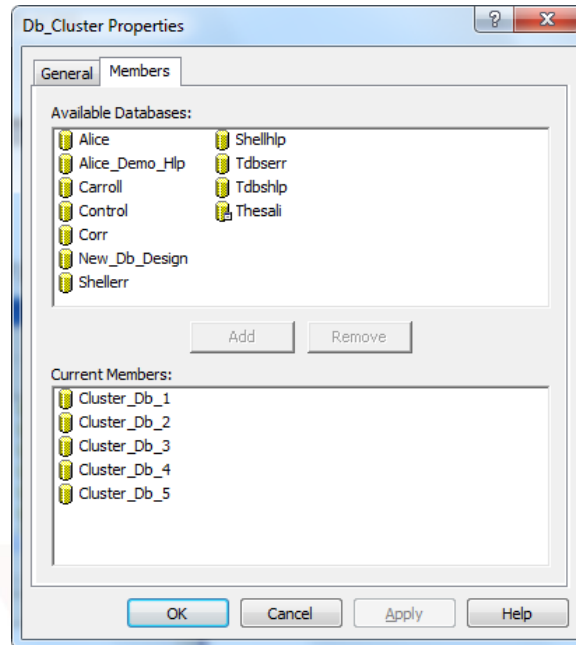
A window will appear showing the cluster properties:



**Figure 2- 30 Cluster General Properties**

The first tab on the cluster properties form is the 'General' tab. This tab displays the cluster name and can also be used to modify the database cluster's description. The name of a database cluster cannot be changed. You can, however, copy the cluster to a new name.

The other tab on the database cluster properties form, is the 'Members' tab:



**Figure 2- 31 Cluster Member Properties**

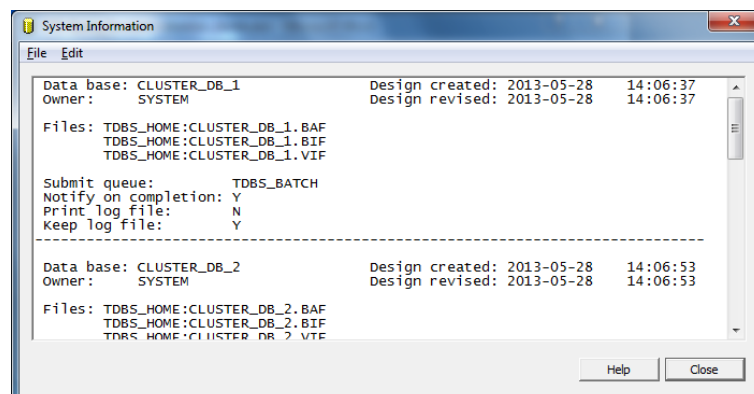
You can modify the cluster member list by adding or removing databases (and / or other clusters) from the 'Current members' list. To add a database select it in the 'Available databases' list and click on the 'Add' button. To remove a database, select it in the 'Current members' list and click on the 'Remove' button. Multiple selections can be made by holding down the <Ctrl> key, whilst clicking on the databases to select.

### Deleting a Cluster

Select the cluster in the mmc window and then select 'Delete' from the action menu. A Yes/No confirmation box will appear before the cluster is removed from TRIP. Deleting a cluster has no effect upon its component databases.

### Related CCL Commands

Once a cluster has been opened, issuing the SStatus command will give the status of all database members of the cluster. However, the SStatus *clustername* command will give the actual status of the cluster, for example:



**Figure 2- 32 A SStatus screen for a cluster database.**

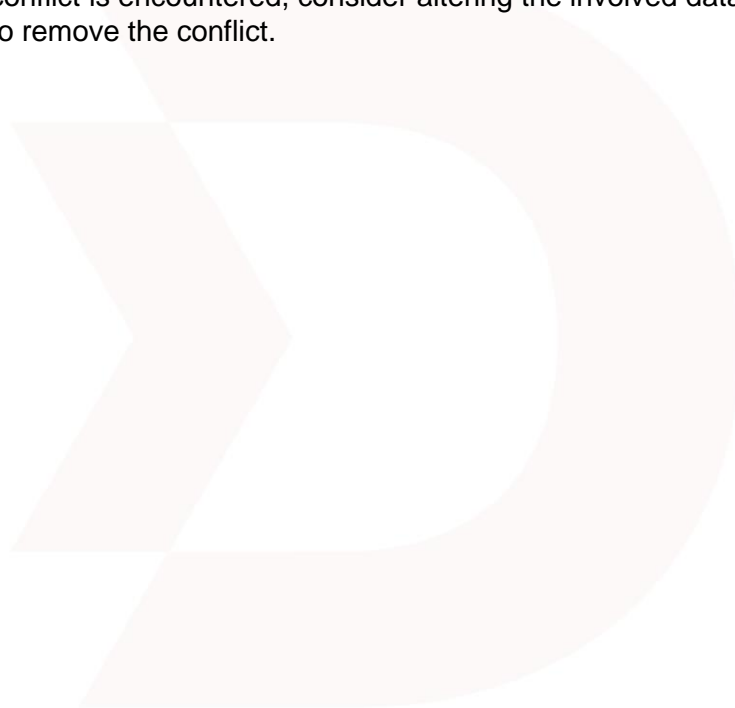
Any database that you have access to can be a member of the cluster. Once a cluster has been created, granting access to it is the same as for any other database and is accomplished through the DB Access menu.

## Single Namespace

TRIP only defines a single namespace to which the names of databases, database clusters, fields, field groups and field views all belong. While TRIP does not prohibit having a field or field group with the same name as a database, such conflicting naming will mean that the conflicting databases cannot be open for search at the same time.

When encountering a naming conflict, close the database(s) in conflict using the CCL command "CLOSE".

Naming conflicts should be avoided as part of database design since such conflicts are likely to result in undesirable application behavior. If a naming conflict is encountered, consider altering the involved database design(s) so to remove the conflict.



## Chapter 3: Thesauri

### What Is a Thesaurus?

A typical thesaurus that one might find on a library bookshelf or packaged with a word processor is often nothing more than a collection of synonyms for a given word or phrase. A structured online thesaurus, however, is more than a single-level assortment of equivalent or synonymous terms. It is a hierarchical, multilevel reference database of terms similar to a manuscript outline that permits vertical as well as horizontal movement among terms and their meanings. This structure allows a user who wishes to research a particular expression to find a broader, more expansive classification for it, a narrower, more divided subterm of it, and terms that are related to it as well as synonyms.

A thesaurus can solve many problems when used while searching a database:

- Finding the correct, accepted or established term for something. This is very useful when a database designer wishes to create a controlled vocabulary using synonyms, and is generally automated using an ASE behind a TRIPclassic data entry form.  
For example, if the only acceptable abbreviation for 'United States' during data entry is 'US', an ASE could be used to validate every insertion made to the box in question via thesaurus. If the data enterer types 'United States', and this term appears as an accepted term in the thesaurus, it is valid. If 'United States' is listed as a synonym of an accepted term, it will be replaced with 'US'.
- Finding more general, 'umbrella' terms to use when the one you were searching with gives no results.  
For example, in a database containing race horse bloodlines and breeding data, the broader or more inclusive term for a champion thoroughbred brood mare called 'Bonney Girl Blue' might be 'Thoroughbred Dam'.
- Finding more limited or precisely defined terms when the one in use produces an enormous number of hits, or a search result consisting of uninteresting generalities.  
For example, in a pharmaceuticals thesaurus, narrower terms or subcategories of the expression 'painkiller' might include 'aspirin', 'acetaminophen', 'ibuprofen', 'codeine', 'morphine' etc.
- Expanding a search laterally by locating terms that are somehow related to or associated with the one being used for searching, that is, using a synonym list or directory. Related terms may or may not be on the same level.  
For example, related terms for 'daisy' might be 'rose', 'bluebell' and 'pansy', or they could be 'flowering plants', 'Shasta daisy', 'roadside weeds' and 'composite flowers'.



- Verifying the usage of a term by searching for its summary or definition. Rather than searching a term and reading its description dictionary-fashion, you could search a term's synopsis for key words and use these to search the database for appropriate expressions. For example, if the definition of 'painkiller' were 'a drug or narcotic that alleviates physical suffering, i.e. pain palliative', a productive search might include the words 'pain' and 'palliative'.

Many of these tasks could be accomplished with a good dictionary. An online thesaurus, however, is faster to use, and its contents are tailored specifically for applications with one or several databases.

## A Simple Thesaurus

A small thesaurus containing terms related to train equipment, types of train cars and the names of individual engines is here diagrammed vertically:

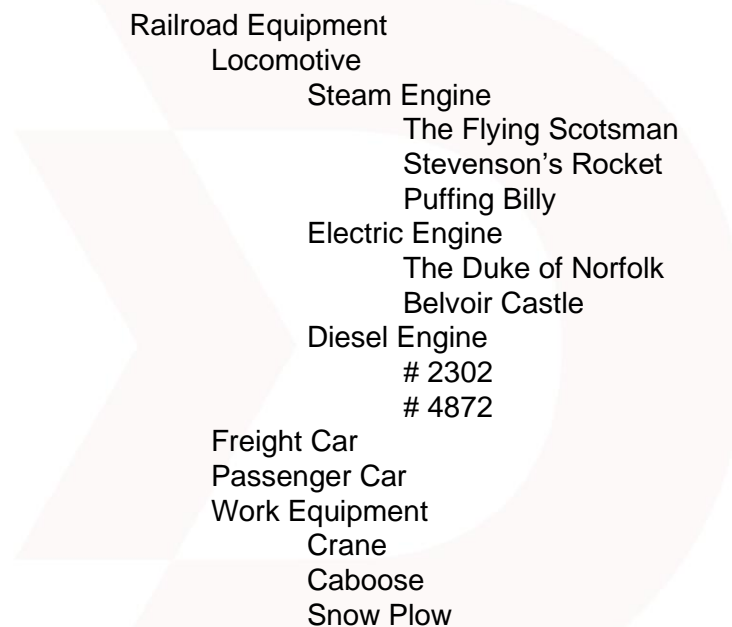


Figure 3–1 The 'Train' thesaurus, vertical representation

and here horizontally:

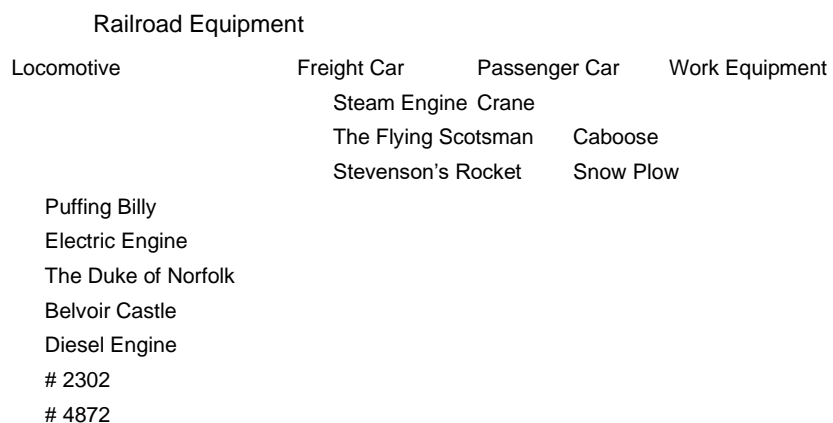


Figure 3–2 The 'Train' thesaurus, horizontal representation

In this example, the immediate sub groupings of the term 'Railroad Equipment' are 'Locomotive', 'Freight Car', 'Passenger Car' and 'Work Equipment', while the next subdivision of 'Locomotive' includes 'Steam Engine', 'Electric Engine' and 'Diesel Engine', and so on.

Seen another way:

Term	Broader Term	Narrower Term(s)
Railroad Equipment	-----	Locomotive, Freight Car, Passenger Car, Work Equipment
Locomotive	Railroad Equipment	Steam Engine, Electric Engine, Diesel Engine
Freight Car	Railroad Equipment	-----
Passenger Car	Railroad Equipment	-----
Work Equipment	Railroad Equipment	Crane, Caboose, Snow Plow
Steam Engine	Locomotive	The Flying Scotsman, Stevenson's Rocket, Puffing Billy
Electric Engine	Locomotive	The Duke of Norfolk, Belvoir Castle
Diesel Engine	Locomotive	# 2302, # 4872
The Flying Scotsman	Steam Engine	-----
Stevenson's Rocket	Steam Engine	-----
Puffing Billy	Steam Engine	-----
The Duke of Norfolk	Electric Engine	-----
Belvoir Castle	Electric Engine	-----
# 2302	Diesel Engine	-----
# 4872	Diesel Engine	-----
Crane	Work Equipment	-----
Caboose	Work Equipment	-----
Snow Plow	Work Equipment	-----

Table 3-1 Record contents and thesaurus design for 'Train'

In this illustration the term 'Railroad Equipment', which has no larger category or broader term, is the top term in the hierarchy. Certain pieces of railroad equipment such as 'Passenger Car' and articles of work equipment such as 'Caboose' are terminal expressions, since they contain no subcategories or narrower terms. The names of individual locomotives such as 'Puffing Billy', 'Belvoir Castle' and '# 4872', which have the greatest number of larger term groupings above them and no narrower terms, are the lowest terms. A terminal expression may or may not be a lowest term in the hierarchy; however a lowest term is always a terminal expression.

A thesaurus can be considered an upside-down 'tree', where the top term is the root and the terminal expressions are the leaves.

There can be more than one tree in a thesaurus, that is, a thesaurus may contain several top terms, each marking the highest level of an individual tree.

## Creating a Thesaurus

Several steps are necessary in building a thesaurus. First and foremost is the conceptual data layout and physical database design, the default conceptual layout being defined by the system.

Next, the planner must decide if the database will be filled manually (using data entry) or automatically (via TForm file). If using data entry, the designer must build a data entry form; if using TForm, he or she must create a program to convert existing online thesaurus data to TForm, or manually create a TForm file via the system editor.

Finally, the defaults should be defined, such as a report, entry form etc.

To create a thesaurus, highlight 'databases' in the mmc window and select 'New' then 'New Thesaurus...' from the action menu.

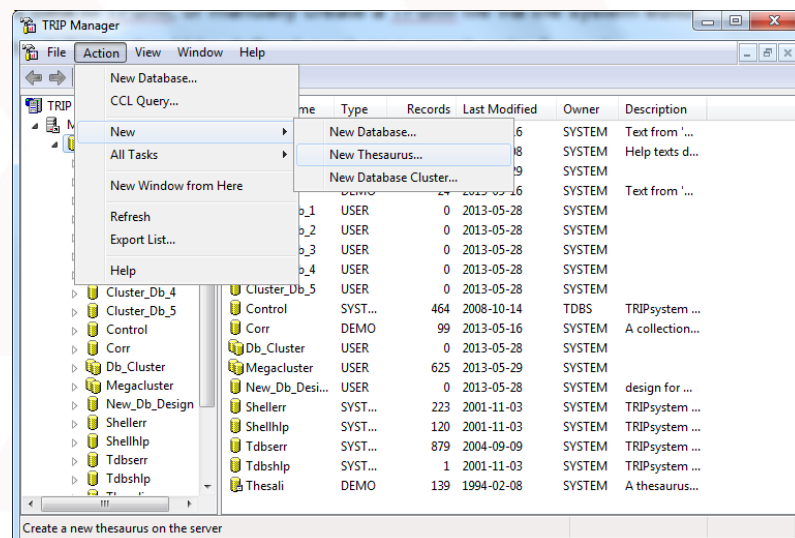


Figure 3–3 New Thesaurus Menu

This will start the New Thesaurus wizard. As this wizard is identical in form to the Database design wizard, refer to the section on creating a new database for more details.

*Note:*

*A newly created thesaurus will contain fields according to the TRIP thesaurus design template.*

## Thesaurus Structure

TRIP provides a template for thesaurus creation, the structure of which is shown below:

Field Acronym	Field Type	Indexed?	Comments
CTX	PHrase	Yes	Controlled Term
BTX	PHrase	No	Broader Term
NTX	PHrase	No	Narrower Term
RTX	PHrase	No	Related Term
UFX	PHrase	Yes	Synonyms
SNX	TExt	Yes	Scope Note/term description
NRX	PHrase	No	Hierarchical position designation used for numerical decimal classification

Table 3–2 The thesaurus template

Note:

*The PHrase field NRX may be used by a data entry facility to contain a number expressing the level of the CT terms in the thesaurus. It is optional, and has no other function.*

## Data Layout

In the following we will use the terms CT, BT and so on, instead of the field names with the added 'X'. That letter is added only because the thesaurus terms are reserved words in CCL and can not be used as field names.

In the hierarchical or conceptual design of a thesaurus, each record contains both a term and its nearest semantic relatives:

Field Acronym	Field Contents	Comment
CT	term	-----
BT	CT's parent	nearest more general term(s)
NT	CT's children	nearest more specific term(s)
RT	other thesaurus CTs that are related to the term	except CT's parent or child. This can be any other relative, or even a term with no ancestor in common with the CT
UF	other non-CT thesaurus terms that are synonyms or near-synonyms of CT	-----
SN	description of CT	-----
NR	hierarchical number	provided for compliance with ANSI thesaurus structure standard; not used by TRIP

Table 3–3 Record contents and thesaurus design

Using the first two levels of the train example illustrated in Figure 3-1 , the hierarchical relationships of the top five terms are outlined below:

Terms					
	Railroad Equipment	Locomotive	Freight Car	Passenger Car	Work Equipment
CT	Railroad Equipment	Locomotive	Freight Car	Passenger Car	Work Equipment
BT	-----	Railroad Equipment	Railroad Equipment	Railroad Equipment	Railroad Equipment
NT	Locomotive, Freight Car, Passenger Car, Work Equipment	Steam Engine, Electric Engine, Diesel Engine	-----	-----	Crane, Caboose, Snow Plow
RT	-----	Freight Car, Passenger Car, Work Equipment, Railroad Equipment	Locomotive, Passenger Car, Work Equipment, Railroad Equipment	Locomotive, Freight Car, Work Equipment, Railroad Equipment	Locomotive, Freight Car, Passenger Car, Railroad Equipment
UF	-----	Engine	Baggage Car, Cargo Carrier	-----	Maintenance Gear
SN	All vehicles used to transport or maintain persons, objects or equipment by rail	The source of power in a railway caravan, operated by steam, electricity or petroleum fuels.	Any rail carrier designed to transport cargo shipments rather than people.	Any rail carrier intended to transport persons rather than cargo.	Any rail car or accessory used primarily for the preservation and restoration of railway equipment and rights-of-way.
NR	1	1.1	1.2	1.3	1.4

Table 3-4 Hierarchical relationships of the 'Train' thesaurus

While the BTX and NTX fields must have content to form the hierarchical structure of the thesaurus, the contents of the RTX, UFX, SNX and NRX fields are discretionary and serve only to make the thesaurus more useful.

A term may be the CT term of more than one record, but only if the BT terms of the records differ, that is, if they are either homonyms or the same term seen from different aspects.

In each record the CT field holds exactly one term; that is, CT may not contain more than one subfield. The number of terms or subfields is not restricted for any other thesaurus PHrase field except BT, whose maximum-single subfield default value can be altered.

CT is the only thesaurus field that can be defined as a record name field.

Each term in a BT, NT or RT field of a record must also appear as the CT of another record, which is why they are not indexed (not searchable) in those records where they are not the CT term.

The UF terms, the synonyms or near-synonyms, must not appear as CT terms with records of their own.

*Note:*

*TRIP will not automatically detect a thesaurus design error wherein term 1A has NT=1B and term 1B has NT=1A. The same is true if term 1A has NT=1. Attempting to Display Down from a level above 1A can then result in a loop and possibly a crash if the Display MAXimum definition is sufficiently large. Care should be exercised to avoid such looping conditions when creating a thesaurus.*

## Thesaurus Database Design

### General Thesaurus Properties

Refer to Chapter Two of this guide, 'Creating a Database' and 'Modifying the Fields Collection' for background information on designing a thesaurus.

### Special Thesaurus Fields

As the CTX field is the only mandatory field, it could be made a record name field with the understanding that the contents of CTX must then be unique in the database. The 'Part Record Field' option is unavailable for thesaurus design.

### Defaults

The default report is the same as for database design.

Be cautious in designing customized reports for use with a search form, as the same report will be used for both Show and thesaurus Display orders. Although it is possible to override the format used to display thesaurus output by designating another default, we strongly suggest that you contact your local TRIP agent for guidance before doing so.

### Character Sets

You may select a character folding class as for an ordinary database.

### Description of the Thesaurus

The thesaurus description appears on SStatus requests as with a standard database.

### Other Thesaurus Properties

The 'ASE (Application Software Exit)', 'Sentences and Paragraphs' and 'Index/Update Submission' screens are identical to those provided for standard database design.

### Field Definition

To create fields in addition to those provided in the thesaurus template or modify the attributes of the pre-existing fields, use the 'Modify Fields Collection' form. Any customised appended fields will not be shown in a Display of the thesaurus, however.

The 'Index' specification for any of the seven predefined thesaurus fields cannot be altered.

Although each field has been provided with a brief description, you may add additional comments and/or restrictions.

## Filling The Thesaurus

### Using TForm

If your thesaurus data exist in a commercially-procured or editor-constructed text file, the file can be converted to TForm, the input format of TRIP and then loaded into TRIP. To do this you should first design the thesaurus on paper, either manually or using a thesaurus maintenance tool, to ensure that all reciprocal terms within the data are correct. You will also need the field numbers of the thesaurus design, which are displayable using a SStatus order after the thesaurus has been created.

See the Appendix in this guide for more information on creating a TForm file.

### Using Data Entry

To fill a thesaurus manually, the thesaurus designer must first build a data entry form containing at least the CTX, BTX and NTX fields, and preferably the other fields as applicable. Data entry then proceeds as usual, constructing term relationships one record at a time. You must enter as many records as there are terms or nodes in the thesaurus tree.

See Chapter Six of the TRIPclassic Manager Guide for more information on building data entry forms.

## Related CCL Commands

Thesaurus designs can be EXPOrted and IMPOrted in the same manner as standard database designs, so that any additional fields that may have been appended to the thesaurus design will be maintained during a move between CONTROL files.

### SStatus

You can use the CCL SStatus command to review general information about your thesaurus once the design has been completed and saved:

```
SStatus databasename ↵
```

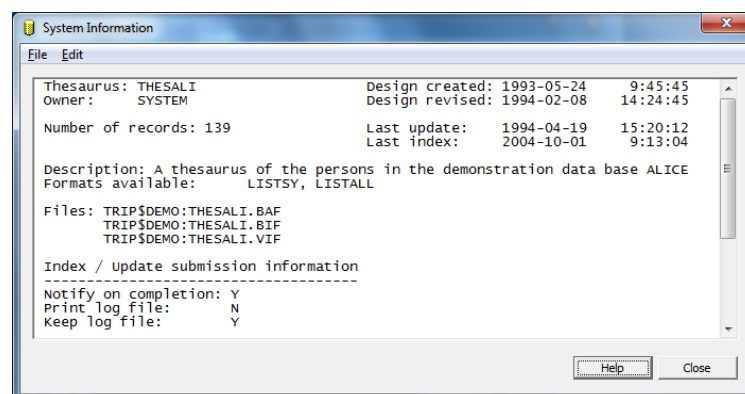


Figure 3-4 SStatus for thesaurus 'Thesali'



## Show

```
Show BASE ↵  
Show BASE LIST ↵
```

## IMPOrt/EXPOrt

```
IMPOrt THEsaurus=thesaurusname file=filename ↵  
EXPOrt THEsaurus=thesaurusname file=filename ↵
```



## Chapter 4: System Logging Functions

### Overview

System logging functions include accounting, auditing and event monitoring.

Auditing is the logging of certain user activities and the time each was performed on the TRIP system. These may include the databases a user has opened and closed, searches performed and output produced.

Accounting is a cost accrual for all records shown or printed by a user according to a predesignated unit cost per field specified in the database design. The costs recorded can be affected using the DEBIT output function (for more information, see the 'Debit' section in Chapter Six, 'Output Format Reference Guide' of this manual).

Event monitoring is a feature that allows the TRIP systems administrator or DBA to output events from TRIP sessions.

### Activating System Accounting Functions

The simplest method of switching on logging functions is to create a blank file called DEBIT.LOG in the directory pointed to by TDBS\_SYS.

If you have a directory for the storage of accounting files defined under the logical name TDBS\_ACCDIR, TRIP will create an accounting file automatically according to the value specified by the logical name TDBS\_ACCFLG. See the next section for more information.

### Assigning Field Costs for Accounting

Assigning a cost to a field when a record is output is done in the database specification. See Chapter Two, 'Database Design' for more information regarding field cost assignment.

### Accounting function Logical Names

There are two logical names that control the user logging functions in TRIP.

The first, TDBS\_ACCDIR defines a directory in which the system accounting logs are kept.

For both UNIX and Windows, TRIP assumes the accounting file is kept in TDBS\_SYS if the logical name TDBS\_ACCDIR is not defined. Any definition of TDBS\_ACCDIR that a user may have set up in his or her own environment will be overridden by any definition in tdbb.conf, which is located in the conf directory of the TRIPsystem installation.

The second logical name, TDBS\_ACCFLG, can be used to customize the name and content of accounting logs.

In UNIX and Windows, any definition of TDBS\_ACCFLG in a user's environment will be overridden by a definition in tdbb.conf.

The value of `TDBS_ACCFLG` is an integer bitmask, whose value varies between 0 and 255. The meaning of each individual bit (0–7) is explained below.

### TDBS\_ACCFLG Bits

To compute the value required for the setting of `TDBS_ACCFLG`, simply add the values of those bits that you wish to enable. For example, to enable all possible logging and activate show focus accounting, set these bits:

Bit Number	Bit Value
3	8 ( $2^3$ )
6	64 ( $2^6$ )

Therefore, the value of `TDBS_ACCFLG` should be  $(64+8)=72$ .

#### Bit 0

This bit (value 1) specifies the use of a user specific account file called `TRIPusername.LOG`, if not otherwise specified by bit number 1. For example, if the user name is 'FRED', then the user-specific accounting log in `TDBS_ACCDIR` will be called 'FRED.LOG'. Any pre-existing `DEBIT.LOG` file will not be used.

#### Bit 1

This bit (value 2 or  $2^1$ ) uses the SIF file name as the name of the account file. If this bit is set, then the name of the user-specific accounting log is the filename portion of the SIF, as specified by the logical name `TDBS_SIF`.

If `TDBS_SIF` is defined as the UNIX path + filename  
`/usr/users/sif_files/jim_johnson.sif` and the username is 'FRED', the accounting file will be called `$TDBS_ACCDIR/jim_johnson.LOG`.

If `TDBS_SIF` is defined as the Windows path + filename  
`C:\users\jjohnson\sif_files\jim_johnson.sif` and the username is 'FRED', the accounting file will be called `%TDBS_ACCDIR%\jim_johnson.LOG`.

*Note:*

*If `TDBS_SIF` does not contain a filename specification, setting this bit will have no effect (see the section entitled 'Logical Names' in Chapter Twelve of this manual for the definitions of `TDBS_SIF`).*

#### Bit 2

This bit (value 4 or  $2^2$ ) uses the SIF file name as the identifier in the account file, rather than the TRIP user name. By default, every entry written to the accounting file contains the TRIP username involved.

Setting this bit instructs TRIP to replace the TRIP username with the filename portion of the `TDBS_SIF` definition.

#### Bit 3

This bit (value 8 or  $2^3$ ) logs all possible information. If this bit is not set, logging will not be performed when a database cluster is opened and when the CCL orders Find, FRequency and MEasure are used.

#### Bit 4

This bit (value 16 or  $2^4$ ) specifies that TRIP should not accumulate database statistics.

The accounting default is for TRIP to accumulate accounting information for all actions taken, and to record this information only when the user logs off the system. If this flag is set, an accounting line is written whenever databases are changed with the BASE command. Statistics will be written for the last database opened upon logout, therefore this flag will be of no use if the user opens only a single database during their entire session.

#### Bit 5

This bit (value 32 or  $2^5$ ) specifies that TRIP should not collect output statistics, but should write accounting information whenever a new Show request is begun.

Setting this bit directs TRIP to record statistics for each Show command separately, so that in the event of an abnormal termination only those statistics for the last Show request performed will be lost.

#### Bit 6

This bit (value 64 or  $2^6$ ) switches accounting on for Show FOCUS. The TRIP default provides accounting information only for normal Show procedures, and does not typically include Show FOCUS.

#### Bit 7

This bit (value 128 or  $2^7$ ) dictates that only records in open databases can be shown. This type of accounting prevents a user from Showing the results of previous searches—to view these, the user must reopen the database against which those searches were made. Print commands for prior searches are still allowed.

#### Bit 10

This bit (value 1024 or  $2^{10}$ ) dictates that the name of the accounting log file is to be the system default DEBIT.LOG, written to the directory as indicated by the TDBS\_ACCDIR logical name. This bit is typically used by itself, resulting in an accounting log with the default contents and name, but stored in a custom directory. This bit is mutually exclusive with bits 0, 1 and 2.

### Accounting Log File Format

The accounting log is a shared sequential file, with each ASCII-readable record containing a maximum of 255 characters. The records or lines have a predefined time of recording, starting and ending position (given in columns) and length, and for convenience are referred to by a single-letter acronym such as B-line.

Each user's session as recorded in the accounting file may contain any combination of the line types shown overleaf:

Line Name	Line Acronym	Starting Position	Ending Position	Length <sup>3</sup>	Field Description	When Recorded
Beginning	B-line <sup>1</sup>	3	50	48	Session identity	At the beginning of a session
Changing or Closing	C-line	52 69	67 88	16 20	Database name Closing date and time	On changing a database or logging out
Exit	E-line <sup>2</sup>	52 72	71 171	20 100	Closing date and time Operating system statistics	On exit from the system
Field	F-line	52 69	67 255	16 187	Database name Copyright holder information	On output of a copyright-protected field
Multibase	M-line	52	253	200	Logs database clusters	Immediately after opening
Opening	O-line	52 69	67 88	16 20	Database name Opening date and time	On opening a database
FreQuency	Q-line	52	253	200	Logs FRequency orders	Immediately after order has been completed
MeasuRe	R-line	52	253	200	Logs MEasure orders	Immediately after order has been completed
Search	S-line	52	253	200	Logs Find orders	Immediately after order has been completed
Usage	U-line	52 69 78 85 94 101	67 76 83 92 99 108	16 8 6 8 6 8	Database name Database connect time (hh:mm:ss) No. records shown (right-justified) Cost of records shown No. records printed (right-justified) Cost of records printed	On exit from the system, but this may be affected by the value of TDBS_ACCFLG

Table 4–1 Line types and the accounting log

<sup>1</sup> For each user entering TRIP, a unique session identity, the B-Line, is created from the date and time of TRIP entry and the operating system user name, the TRIP user name or the SIF file name, depending on the setting of TDBS\_ACCFLG. This session identity is repeated on each line of the debit file in character positions three to fifty, so that each user's entries can be grouped using a simple sort.

<sup>2</sup> The operating system statistics captured by the E-Line include total TRIP connect time and CPU time.

<sup>3</sup> Database names longer than 16 bytes will be truncated and the following blank space replaced by an asterisk “\*”.

### Example

The following example is a log file from a short single-user TRIP session. The initial ten lines of code of the interactive user session reproduced below (Table 4–2) generate the first fourteen lines of output in the accounting file (Figure 4–1). The last line of code, the Print statement, executes a batch job that produces the remaining nine lines of accounting output.

The user name for the session is JANNE, and the TRIP user name is SYSTEM. A cost has been assigned to one of the fields of the database CORR, but no copyright holder has been specified (that information is not mandatory). Bit number three of TDBS\_ACCFLG has been set.

The session consisted of the following TRIP orders:

Timepoint	CCL Command
User's Interactive Session	base corr
	F \$rip
	show
	base alco=alice,corr
	find alice
	show
	show reverse
	freq rname
	meas chaptnr
	f jabber\$ or jan
Batch Print Job	print file=x

The accounting log file would appear as follows:

COLUMN POSITION							
1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80
----- Session Identity ----- ----- ----- ----- ----- ----- ----- -----							
3			51		69	78	84/85 94 101 ...
B JANNE SYSTEM 11-JAN-1991 09:27:15							
O	JANNE SYSTEM 11-JAN-1991 09:27:15		CORR		11-JAN-1991 09:27:20		
S	JANNE SYSTEM 11-JAN-1991 09:27:15		Find \$rip				110
M	JANNE SYSTEM 11-JAN-1991 09:27:15		BASE alco=alice, corr				
O	JANNE SYSTEM 11-JAN-1991 09:27:15		ALICE		11-JAN-1991 09:27:30		
S	JANNE SYSTEM 11-JAN-1991 09:27:15		Find alice				
Q	JANNE SYSTEM 11-JAN-1991 09:27:15		freq rname				
R	JANNE SYSTEM 11-JAN-1991 09:27:15		meas chaptnr				
S	JANNE SYSTEM 11-JAN-1991 09:27:15		Find jabber\$ OR jan				
C	JANNE SYSTEM 11-JAN-1991 09:27:15		CORR		11-JAN-1991 09:29:20		
C	JANNE SYSTEM 11-JAN-1991 09:27:15		ALICE		11-JAN-1991 09:29:20		
U	JANNE SYSTEM 11-JAN-1991 09:27:15		CORR	00:02:02 2	0	0	0
U	JANNE SYSTEM 11-JAN-1991 09:27:15		ALICE	00:01:49 2	0	0	0
E	JANNE SYSTEM 11-JAN-1991 09:27:15		11-JAN-1991 09:29:23 ELAPSED: 0 00:02:07.75 ...				
... CPU: 0:00:07.76 BUFIO: 08 DIRIO: 71 FAULTS: 56							
B JANNE SYSTEM 11-JAN-1991 09:30:22							
M	JANNE SYSTEM 11-JAN-1991 09:30:22		BASE alco=alice, corr				
O	JANNE SYSTEM 11-JAN-1991 09:30:22		CORR		11-JAN-1991 09:30:20		
O	JANNE SYSTEM 11-JAN-1991 09:30:22		ALICE		11-JAN-1991 09:30:20		
C	JANNE SYSTEM 11-JAN-1991 09:30:22		CORR		11-JAN-1991 09:30:20		
C	JANNE SYSTEM 11-JAN-1991 09:30:22		ALICE		11-JAN-1991 09:30:20		
U	JANNE SYSTEM 11-JAN-1991 09:30:22		CORR	00:00:04 0	0	17	0
U	JANNE SYSTEM 11-JAN-1991 09:30:22		ALICE	00:00:04 0	0	1	0
E	JANNE SYSTEM 11-JAN-1991 09:30:22		11-JAN-1991 09:30:27 ELAPSED: 0 00:00:05.58 ...				
... CPU: 0:00:03.50 BUFIO: 19 DIRIO: 71 FAULTS: 397							

Table 4–2 A sample accounting file

## Event logging

An event logger library has been added to TRIPsystem. This is able to detect various events in TRIP sessions and log its findings to file.

### Overview

Event monitoring is a feature that allows the TRIP systems administrator or DBA to output events from TRIP sessions. If event monitoring is enabled, each session gets its own event log file.

Events in this context are:

- Errors in the current session
- Changes to users (e.g. created, deleted)
- Changes and actions on databases (e.g. created, deleted, opened, closed)
- Submitted batch jobs (index jobs, print jobs and global updates)
- Session changes (login, logout)

## How to Enable Event Logging

Event logging is disabled by default. It is enabled by adding the following property to the `tdbs.conf` file:

Property name	Value	description
<code>TDBS_MONITOR_LIB</code>		Fully qualified path to the monitor library file

It is also possible to generate performance measurements on query executions. This is not enabled by default, even if event monitoring is otherwise enabled, but may be enabled by setting the following property in the `tdbs.conf` file:

Property name	Value	description
<code>TDBS_MONITOR_QPERF</code>		“Y” or “1” to enable query performance event logging

Example, with query monitoring enabled:

```
TDBS_MONITOR_LIB=${TDBS_HOME}/bin/libmonlog.so  
TDBS_MONITOR_QPERF=Y
```

Monitoring is always enabled if the `TDBS_MONITOR_LIB` property is defined and refers to the `libmonlog` shared object or DLL file.

## Parameters

Additional parameters to the event monitor can be given by adding the following properties to the `tdbs.conf` file:

### **TDBS\_MONLOG\_FLUSH**

Determines if log statements should be forced to disk immediately as they are written, or allowed to delay in the file system cache.

Enable by specifying Y. Default is N (false).

### **TDBS\_MONLOG\_MONOLITHIC**

Determines if the monitoring event log is monolithic or per session. When monolithic logging is enabled, all sessions share the same log file.

Default is Y (true). Disable by specifying N.

### **TDBS\_MONLOG\_MONOLITHIC\_PERIOD**

Determines the time period of monolithic logs. A new log file is created for each new period. Valid values:

- HOUR
- DAY
- WEEK
- MONTH
- YEAR

Default is DAY.



## TDBS\_MONLOG\_TSTAMP

Determines if a year-to-second time stamp should be included in the log file for each log row.

Default is N (false). Enable by specifying Y.

## Event Log Output

The format of the event log file is a comma-separated values (CSV) file. The following tables describe the fields in the file.

Common fields for all message types:

Field nr	Value Type	Description
1	Process ID	The process ID of the process from which the event originated. This is typically the TRIP Daemon (tripd), the TRIPserver (tbserver), and TRIP classic (trip), but also includes serverside TRIP processes that directly use the TRIP kernel.
2	Time stamp	Time stamp in the format “YYYY-MM-DD hh:mm:ss”. This field contains an empty value unless the TDBS_MONLOG_TSTAMP property is set to Y.
3	Hi-res time	A nano-second time offset from the start of the process from which the event originated. The format of this value is hh:mm:ss:mmm.uuu.nnn.
4	Message Type	Indicates the type of the information on the current row. The value types of the remainder of the fields are determined by this value.

The following fields apply to message type PROCINFO, which denotes basic process information. This information is sent when the monitored process is starting up.

Field nr	Value	Type Description
5	Process Type	The type of the process. This is “TRIP kernel” for a TRIP session or “TRIP daemon” for the TRIP Daemon (tripd).
6	User name	The name of the currently logged on user
7	Is Session	Indicates if the process currently hosts an active TRIP session, i.e. a logged on user.

The following fields apply to message type EVENT, the most common entry in the event monitoring log.

Field nr	Value Type	Description
8	Severity	The severity of the event. Possible values are INFO, WARNING, ERROR and CRITICAL.
9	Resource Type	The type of resource associated with the event. Typical values are USER and DATABASE. Additional possible values are SESSION, HOST, PROCESS and JOB.
10	Resource ID	The identity name of the resource associated with the event.
11	Event Type	The type of the event being signalled. Valid values are: <ul style="list-style-type: none"> <li>STARTED – e.g. a successful user login</li> <li>STOPPED – e.g. a logout</li> <li>OCCURRED – a generic event</li> <li>CREATED – e.g. a database was created</li> <li>DESTROYED – e.g. a database was deleted</li> <li>ERROR – e.g. an error has been raised</li> <li>WARNING – e.g. a non-critical error has occurred</li> <li>FAILED – e.g. generic non-erroneous failure</li> <li>MEASUREMENT – e.g. query performance</li> </ul>
12	Event Name	Display name or title for the event. In case of errors and warnings, this indicates the code of the TRIP error.
13	Description	Human-readable event description. In case of errors and warnings, this is typically the TRIP error message.

## Log File Location and Name

The event log is written to the directory indicated by the TDBS\_LOG property in the tdbb.conf file.

The name of the file is determined by whether monolithic mode is enabled (see the description of the property TDBS\_MONLOG\_MONOLITHIC) and the period of the monolithic logging (see the description of the property TDBS\_MONLOG\_MONOLITHIC\_PERIOD).

If monolithic logging is disabled, the log file name will be:

```
eventlog_<year><month><day><hour><min><sec><pid>.log
```

If monolithic logging is enabled, the log file names depend on the time period:

- HOUR `eventlog_<year><month><day><hour><min><sec>.log`
- DAY `eventlog_<year><month><day>.log`
- WEEK `eventlog_<year><dayofyear>.log`
- MONTH `eventlog_<year><month>.log`
- YEAR `eventlog_<year>.log`

Year is always a 4-digit number. The values for month, day, hour, minute and second are all zero-padded 2-digit numbers. The value for 'dayofyear' used for WEEK time period is a zero-padded 3-digit number that denotes the day of the year for the Monday in the current week.



## Part 2:

## Forms



## Chapter 5: TRIPclassic Data Entry Forms

Data entry is the process of manual insertion of data into a database. To do this, the database manager or a user with access to the database designs a screen form giving access to the fields of the database.

A data entry form consists of one or more screen pages for the entering of the records. Headers, frames and visual characteristics of different kinds may be added to the form. The data entry form can be used both for entering new records and editing old ones. The records created using the form are added directly to the BAF file.

To view the existing data entry forms for a given database, expand its sub-tree in the mmc window then click on the 'Entry Forms' icon. A list of entry forms will then appear.

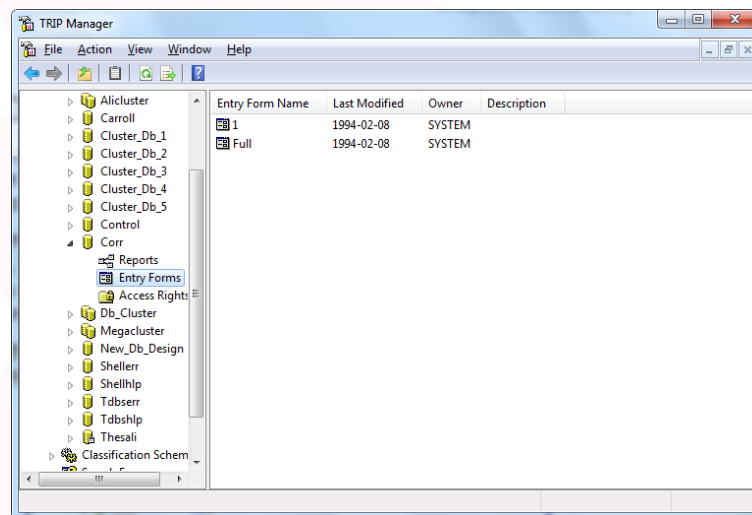


Figure 5–1 Entry forms for database CORR

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

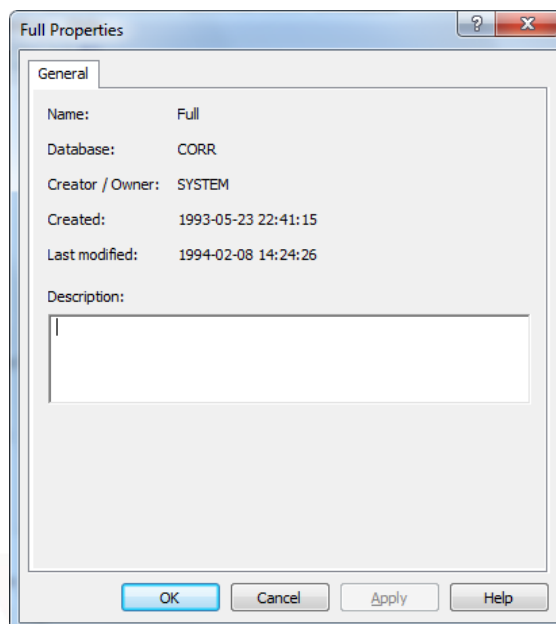


Figure 5–2 Properties for CORR entry form FULL

## Creating and Modifying TRIPclassic Data Entry Forms

TRIPmanager currently has no means of carrying out these operations. It is hoped to include this functionality in a later release. For now, consult the TRIPclassic user guide for details of how to perform these actions.

## Copying TRIPclassic Data Entry Forms

To copy a data entry form, click on 'Entry Forms' in the chosen database sub-tree, select the form to copy and select 'Copy' from the action menu.

Next click in the right hand pane of the mmc to deselect the Entry Form being copied, then select 'Paste' from the action menu.

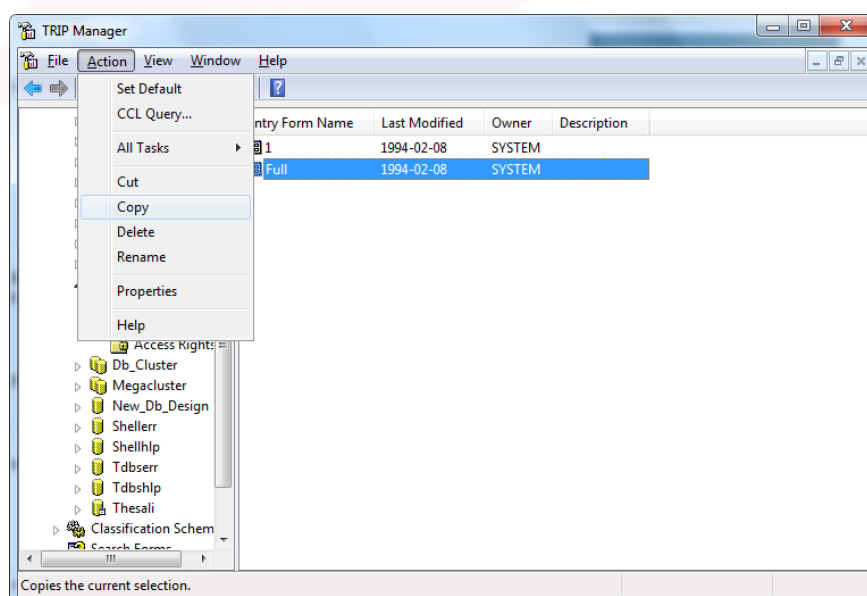


Figure 5–3 Copy a Data Entry form

A dialogue will appear, in which you may enter the name for the new Entry form copy and click on the 'OK' button to confirm the action.

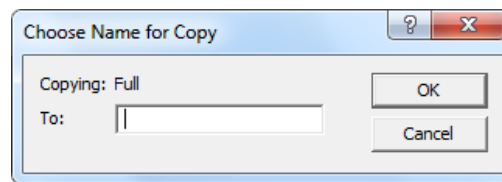


Figure 5-4 Name New Data Entry Copy

a new Entry Form creation confirmation will then appear.

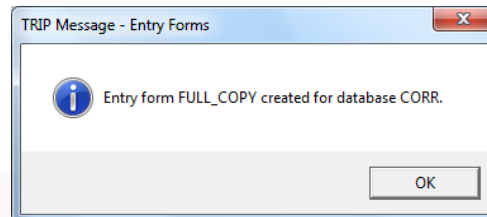


Figure 5-5 Data Entry Copy Confirmation

Clicking on the 'OK' button will clear the confirmation. The copy of the new form has now been created.

## Deleting TRIPclassic Data Entry Forms

To delete a data entry form, click on 'Entry Forms' in the chosen database sub-tree, select the form to erase and select 'Delete' from the action menu.

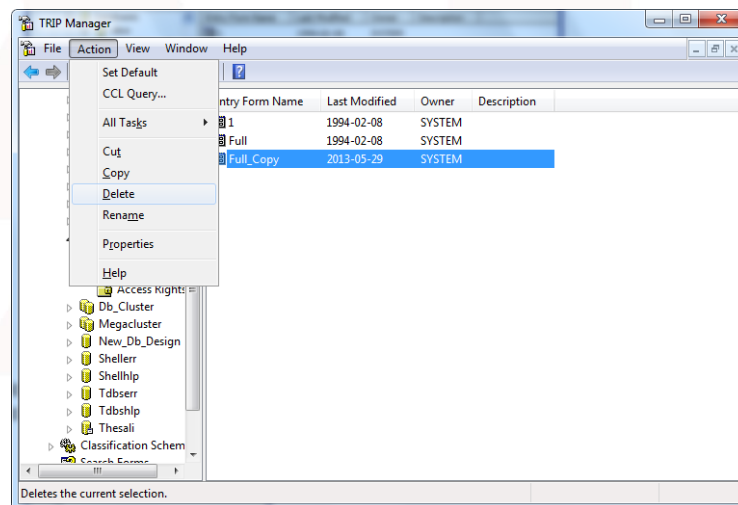
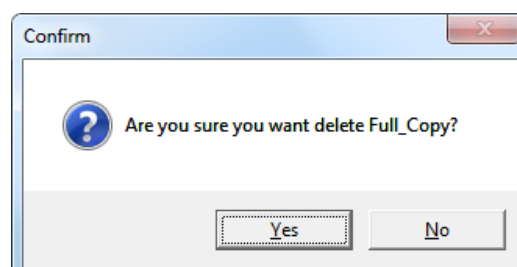


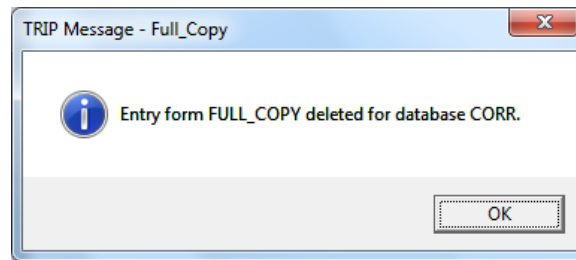
Figure 5-6 Delete a Data Entry form

A 'Yes/No' confirmation dialogue will appear. Click the 'Yes' button to confirm the action, or the 'No' button to cancel it.



**Figure 5–7 Delete Data Entry form confirmation**

Clicking on 'Yes' will cause a deletion confirmation to appear.



**Figure 5–8 Data Entry form Deleted**

Clicking on the 'OK' button will clear the confirmation. The selected data entry form has now been deleted.



## Chapter 6: Reports / Output Formats

### Notes:

- *In keeping with modern database terminology, TRIP's output formats are often now referred to as 'reports'. However, in reality, persistence in using the original title 'output format' means that the two terms are, in effect, interchangeable.*
- *Due to TRIPclassic's legacy of having been designed in the era of 'dumb' terminals (e.g. the venerable VT100), which all utilised fixed width fonts arranged in columns and rows, many of the character positioning functions used when designing TRIP reports for textual output are based on an imaginary, user pre-defined, screen grid with each square on the grid containing one character. Calculations as to text and output box positions, therefore, relate to the dimensions of this underlying grid; a fact which should be kept in mind when reading this section.*
- *In contrast to designing textual output reports, when designing reports for (e.g.) HTML, or XML outputting, the report designer is free to include whatever tags they wish and the screen grid can largely be considered an irrelevance.*

A report, or output format, specifies how a record in a database is to be output. It produces a user-specific data summary containing only the pieces of information required by the user, in the blueprint or configuration most useful to him or her. Using specialised symbols and functions, a user may specify which parts of the record should be output, how the different parts should be separated, in what position on the page (screen or printed page) they should be written, and what additional information should accompany them.

Any user with read-access to a database may create such a tailored report and send it either to screen or printer, but only the author of a report or the database administrator (DBA) may edit or delete it.

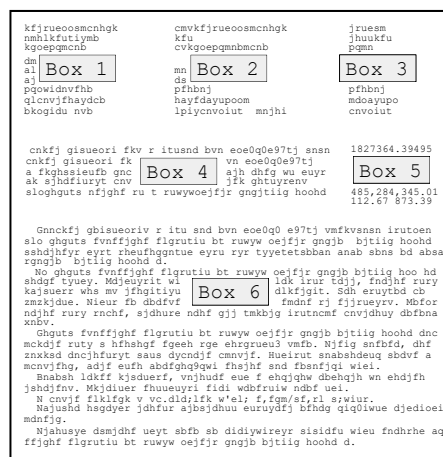
All specifications (with the exception of text string literals) are case-insensitive.

### The Report

A report specifies how a single record in a database should look when printed, which TRIP uses as a template to print one record after another. A report consists of the left chevron or less-than symbol [<], one or more layout box definitions and the right chevron or greater-than symbol [>].

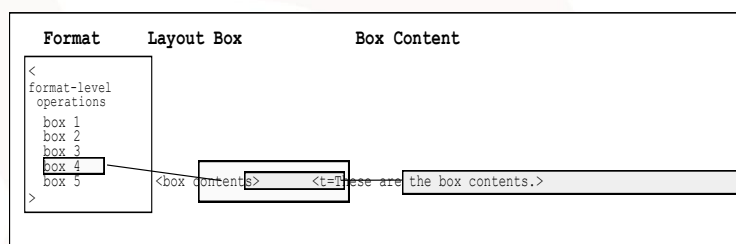
Since records vary in length, an output page may not be identical to the previous one.

We recommend that you diagram one page of your report before attempting to create the code, so that you have some idea of how it will (or should) look:



**Figure 6-1 Report layout and construction**

To produce the desired layout, a report is divided into boxes, which in turn contain the items to be printed, as illustrated below.



**Figure 6-2 Report components**

The basic element of a report specification is the box, a block of data that may be placed anywhere on the screen or printed page. Each box consists of a box definition, comprised of one left chevron, the word 'box', the contents of the box and one right chevron, all positioned somewhere within the format definition.

Boxes may contain any or all of the following constituents:

- contents of one or more fields
- field headers, separators and trailers
- text string insertions and constants
- functions and filters
- control characters

Report elements appear in a particular order within the report:

1. The elements of the report specification must be inside the delimiters of a box or box group. There are three exceptions: text variable declarations, a specification of page size (for Print statements), and <Sortfields> must all appear immediately after the 'Begin Format Specification' marker.

2. Headers, separators, trailers, <For> constructs and control variable declarations are the only elements that can be placed outside a box, after the 'Begin Box Group' delimiter.
3. The elements of a format specification can be sorted into two groups, let us call them A and B. Group A contains the output control elements, or headers, separators, trailers, control variable declarations and free-standing functions such as <Indent> and <Noorig>. Group B encompasses box content, including text inserts and names signalling output of field contents (field and data type names).
4. Within a box definition, all members belonging to group A appear before everything that belongs to group B. While the internal order of the elements in group A has no affect on the output, the internal order of the elements in group B will determine the order of the output of the groups.
5. A format is read one line at a time during system processing, therefore it is not possible to refer to a box or variable that has not yet been defined. Such a reference will generate an error message when you create (attempt to save) the format.

## Copying Reports

It is often labour-saving to base new reports on the structure of similar ones. If you need to adapt formats to a standard, you can do that easily using the same technique described in the "Copying TRIPclassic Data Entry Forms" section of the "Forms" chapter in this manual.

*Notes:*

- *At the moment, if you wish to copy a report from one database to another using TRIPmanager, you will get an error if the source report references fields that do not exist in the target database.*
- *As a workaround, you can either use TRIPclassic to perform this action (See the TRIPclassic Administration guide), or you must first create a new report in the target database (See "Creating a New Report", below), then edit the source report and copy-past it's contents into the new target report's open editor, ensuring that any field name issues are resolved before saving the new report.*

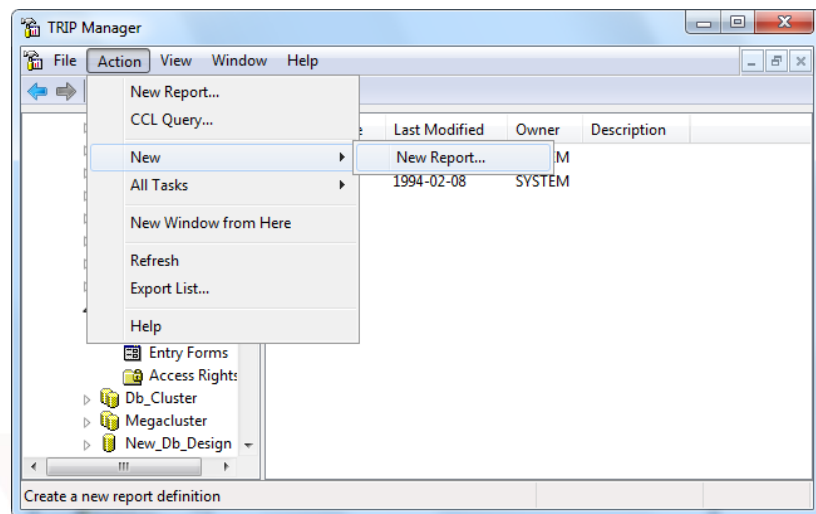
## Deleting Reports

A format that is no longer used should be deleted. This can be done by selecting the report (or reports) to be removed, then using the Delete option in the Action menu of TRIPmanager.

## Creating a New Report

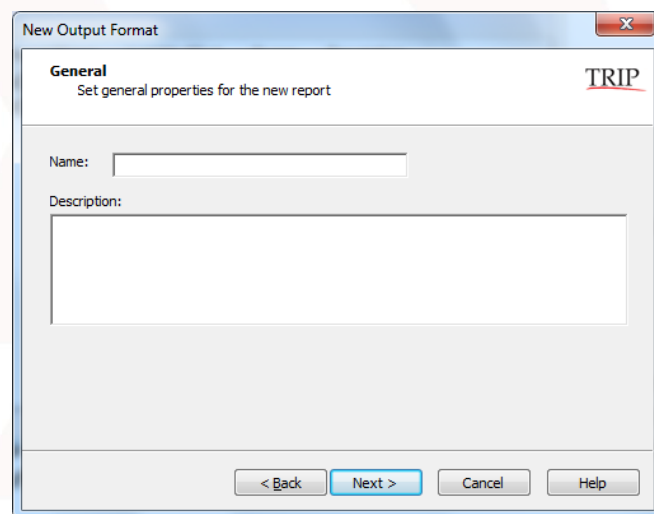
After creating a default report for a database, you should select the default report name on the database design properties, 'General' tab, using the 'Default report' selection box.

To create the default report, click on the plus (+) sign in the mmc window next to the desired database in order to expand its sub-tree, select the 'Reports' sub tree branch, then select 'New' and 'New Report...' from the 'Action' menu.



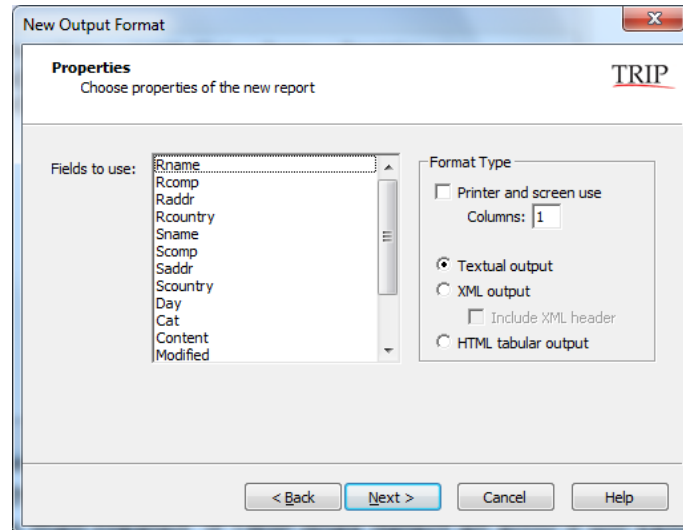
**Figure 6–3 New Report Menu**

You will then be brought into the welcome page of the New Report Wizard. Clicking on <Next> will take you to the General information entry dialog:



**Figure 6–4 New Output Format name entry dialog**

Here you can enter a name and a description for your new output format. When you have done so, click on <Next> to proceed to the new report Properties entry form:



**Figure 6–5 New report Properties dialog**

The Properties dialog, in *Figure 6–5*, short-circuits some of the basic report setup steps that need to be performed every time you create a new report. As can be seen, the dialog has two main areas:

In the left hand area, you can <Shift> or <control> click to select multiple fields to be included in your report.

In the right hand area, the following options are available for creating different format types:

- "Printer and screen use" sets the report up to specify a page size.
  - If you specify a number of columns, the page size is divided by that number.
- Choosing "Textual output" will generate a report something that looks a lot like the TRIP DUMP format

*Note:*

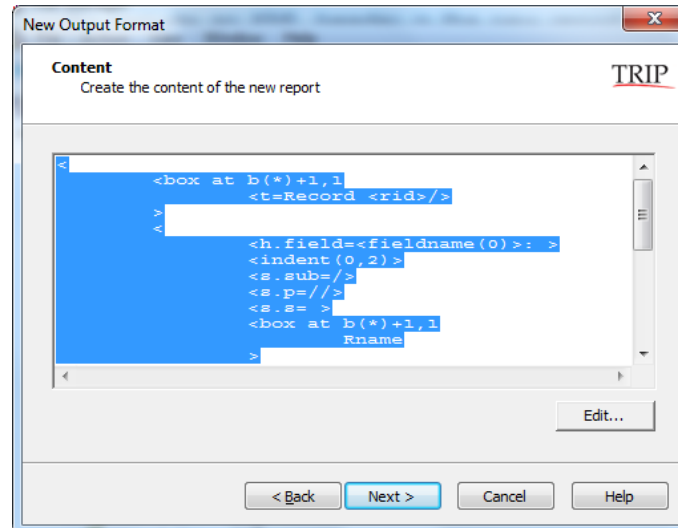
*You can use the CCL command **Show Format=DUMP** to see an example of the TRIP DUMP format.*

- Choosing "XML output", will generate a simple, well-formed, XML document.
  - To include an XML header in the new report (e.g. "<?xml encoding='...' ?>", etc.), select the "Include XML header" checkbox.
- Choosing "HTML tabular output", will generate a simple HTML table similar in layout to the TRIP DUMP format, where each field name and contents generates an individual table cell in the report definition.

*Note:*

*If any of the fields chosen are part fields, a part loop variable is created to control the output of the various components.*

Clicking on <Next> will bring you to the dialog previewing the report's content prior to it actually being committed to the server:



**Figure 6–6 New report Content dialog**

Clicking on <Edit> will bring you into the default external text editor which, for Windows, is 'Notepad'. This can be used to create and modify the specification file for the report. When this file is saved, TRIP checks the new or altered format for syntax errors and, providing there are none, the format is then created. If TRIP does detect an error it will send a message detailing where and what the problem is, and you can re-edit the file as necessary.

## Defining Layout Boxes

### Simple Boxes

The simplest format is one that contains only one box, and the simplest box instructs TRIP to output the contents of a single field. The contents of a particular field are output by including the name of the field in the box.

This example,

```
<<box content>>
```

which we will call 'Corr\_Out', writes only the data from field 'Content' of the demonstration database Corr, beginning on the first free line and in the first empty column on the screen or page. The first record prints in the top and leftmost position on the page, and subsequent records are written on the line immediately after the preceding record with no intermediate linefeeds.

The outer symbols [<] and [>] mark the beginning and end of the report specification (format definition enclosures), while the inner [<] and [>] symbols identify the start and stop of the box description (box definition enclosures).

If the following statements were entered in the CCL command window:

```
BASE corr ↵
Find R=3,5,6 ↵
Show format=corr_out ↵
```

the output would look like this:

```
TELEX NO 312/7
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM
```

PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW  
AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE PRICE? DOES THE  
SYSTEM EXIST ON OTHER MACHINES?  
RON SMITH, SPARKLER INSTITUTE

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,  
called TDBS, is under development. A first prototype will be  
at hand this summer. I will mail further information.

Best regards,

Mats G. Lindquist

Sirs,

I would be grateful if you could send me any information available  
on the free-text retrieval system 3RIP, marketed by your company.

Thank you.

Sincerely,

George Hodge

To make a report specification easier to read, we recommend the use of spaces, tabs and linefeeds to align box and box group definitions. These are editing helps only and do not affect the final appearance of the output.

Adding spaces, linefeeds and comments to improve comprehension and readability, the report outlined above might look like this:

```
!      This is a report:
<
<box content>
>
```

This format may be summarized as follows:

Component	Explanation
! This is an...	Comment
<	Begin format specification
<box content>	Begin box specification, define box, print contents of field 'Content' and end format specification
>	End format specification

This specification will work exactly as the single-line specification given previously, as the spaces and linefeeds will not affect the output. The exclamation mark [!] signifies a comment, and any text from the [!] to the end of the line will be ignored.

For maximum readability, if a box contains only a single item place the 'Begin Box Specification', box definition, box content and 'End Box Specification' on the same line.

If the box contains more than one item, place the 'Begin Box' chevron and box definition on one line, each content item on its own line and indented slightly and the 'End Box' symbol on the line below the last item, aligned with the 'Begin Box' mark:

```
<
<box
  sname
  scomp
  saddr
  scountry
>
>
```

As this is a very simple report lacking defined linefeeds and separators, the output for record numbers three, five and six looks like this:

```
Mr. Ron SmithThe Sparkler Institutel6 Sparkling RoadSparkletownUSA
Mats G. LindquistPARALOG ABBox 2284103 17 STOCKHOLMSverige
George HodgeThe Response ProjectP.O. Box 53DallasTexas 75265USA
```

**Note:**

*The string `tstamp` is used to output the timestamp of a record (the date and time when the record was created or updated). It is treated as a field name and accepts headers, separators etc. like a field.*

### Output of Specific Field Elements

To output the entire contents of a box, use only the field name in the box specification. To output only part of a non-TEXT field, use the field name followed by a full stop and the number of the element you wish to print. For example, to write only the second subfield of the Corr field `sname` you would use

```
<<box sname.2>>
```

Similarly, to print a particular sentence from a certain paragraph of a TEXT field use the field name, a full stop, the paragraph number, full stop and the number of the sentence. This example,

```
<<box content.2.1>>
```

outputs the first sentence of the second paragraph of field Content for each record, and

```
<<box content.*.1>>
```

outputs the first sentence of every paragraph.

Record components are numbered, the head record being zero and the parts numbered from one upwards. To output only a certain field in a part record for each part record, use a full stop, the part number or index and the field name. Using the field `Txt` from the demonstration database Carroll, this example outputs the first sentence, second paragraph, fourth part:

```
<<box .4.txt.2.1>>
```



### Box Numbering

To prevent confusion, it is always preferable to use directed box specifications rather than ambiguous or non-specific statements, therefore we recommend numbering each box consecutively as it is defined. The previous example with box numbering looks like this:

```
!      This is a report:
<
<box 1 content>
>
```

in which the first box to be defined in this format is Box Number 1. Again, the output is the same as the previous <<Box content>> example.

### Box Positioning

The simple format example above prints the data from one 'Content' field after another, without intervening lines to mark the beginning and end of the records. This produces 'run-on data', which can hinder readability.

One way to create record boundaries in reports is through text box positioning on the page, which defines where the upper left-hand corner of each box should be placed.

A layout location may be given in several ways:

- directly, by giving the number of the line and the column (page coordinates) relative to the current record
- indirectly, by referring to the top or bottom line or right- or left-most column of a pre-existing or preceding box, which may be either the last box written or a box identified by a number.
- Absolute and relative positions may be combined within a format in any way you find suitable. Both line and column can be given either as a relative position or an absolute position.

### Positioning Using Coordinates

With reports, any statement using line and column positioning or absolute placement refers to the last written line of the preceding record (Line Number 0), not to a line on the screen or printed page. When you do specify a coordinate position you must give both line and column, separated by a comma.

If the 'Box content' example is rewritten using coordinates like this:

```
<
<box 1 at 2,10 content>
<box 2 rname>
>
```

Box 1 will begin on the second line (the last written line of the preceding record being line 0) and in the tenth column of the screen or paper page. As no coordinates were provided for Box 2, it will appear on the next free line in column one with no separators.

If the format is used to output records three, five and six of database Corr as before, the output will look like this:

```
TELEX NO 312/7
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM
PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW
AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE PRICE? DOES THE
SYSTEM EXIST ON OTHER MACHINES?
RON SMITH, SPARKLER INSTITUTE

Mats G. LindquistMats Lofstrom

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,
called TDBS, is under development. A first prototype will be
at hand this summer. I will mail further information.

Best regards,

Mats G. Lindquist

Mr. Ron Smith

Sirs,

I would be grateful if you could send me any information available
on the free-text retrieval system 3RIP, marketed by your company.

Thank you.

Sincerely,

George Hodge
```

The information contained in the 'Content' field of record number three will begin printing on line two, column ten. When all 'Content' data from record three has been output, TRIP will descend two lines from the last line written, move to column ten and begin printing the output from record five, and so on until all records have been output.

If no box position is provided, the box will start printing in the first column of the first empty line by default. We recommend using box numbering and positional 'at' statements wherever possible.

### Positioning Using Preceding Boxes

Building on the previous example, the initial line position for Box 1 appears below in relation to the bottom line of the last box printed:

```
<
<box 1 at b(*)+2,2 content>
>
```

This specification can be summarized as follows:

Component	Explanation
<	Begin format specification
<box 1	Begin box specification and define Box 1
at b(*)	Position cursor on bottom line of last box written
+2,2	Add two linefeeds and move to column two of the new line
content>	Print contents of field 'Content', each line beginning in column two; end box specification
>	End format specification

and the output:

```

TELEX NO 312/7

ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM

PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW
AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE PRICE? DOES THE
SYSTEM EXIST ON OTHER MACHINES?

RON SMITH, SPARKLER INSTITUTE

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,
called TDBS, is under development. A first prototype will be
at hand this summer. I will mail further information.

Best regards,

Mats G. Lindquist

Sirs,

I would be grateful if you could send me any information available
on the free-text retrieval system 3RIP, marketed by your company.
Thank you.

Sincerely,

George Hodge

```

In addition to the bottom line, you can also use the top line (T) or the rightmost (R) or leftmost (L) columns of another box as reference points. You must refer to a numbered box when using T, R, or L.

Some examples follow overleaf:

Box at T(1),41 column 41.	The box is placed at the top line of box number 1, in column 41.
Box at T(1),R(1)+2	The box is placed at the top line of box number 1, two columns to the right of it.
Box at 10, L(2)	The box is placed on line 10, starting in the same left column of box number 2.

As with box positioning using coordinates, if no box position is provided the box will start printing in the first column of the first empty line by default. We again recommend using box numbering and positional 'at' statements wherever possible.

### Box Proportions

The dimensions of a box may be defined using any of the following:

- number of lines and columns
- number of lines, with columns unspecified
- number of columns, with lines unspecified
- neither lines nor columns specified.

If the field contents of a box occupy less room than has been provided in the box definition, the text block will be padded with empty lines and spaces as necessary.

### Proportioning With Lines and Columns

The height and width of a box can be simultaneously defined by using the 'Size' directive and providing both the number of lines to be output and the line length in characters, separated by an asterisk [\*, pronounced 'by']. For example:

```
<
  <box 1 at b(*)+2,2 content>
  <box 2 at b(1)+1,40 size 2*20 rname>
>
```

outputs box number two as a two line by twenty column block, beginning in column forty of the line below the final line printed for box number one as shown on the next page:

TELEX NO 312/7  
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM  
PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW  
AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE PRICE? DOES THE  
SYSTEM EXIST ON OTHER MACHINES?  
RON SMITH, SPARKLER INSTITUTE

Mats G. Lindquist  
Mats Löfström

Dear Mr. Smith,  
  
3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,  
called TDBS, is under development. A first prototype will be  
at hand this summer. I will mail further information.  
Best regards,  
  
Mats G. Lindquist

Mr. Ron Smith  
[ ]

Sirs,  
  
I would be grateful if you could send me any information available  
on the free-text retrieval system 3RIP, marketed by your company.  
Thank you.

Sincerely,  
  
George Hodge

We will use empty brackets [ ] as a convention in output examples such as the one above to indicate empty lines inserted by TRIP.

If both line and column figures are given there must be no space between each total and the asterisk.

### Proportioning With Lines Only

To define only the vertical or top-to-bottom box size, give only the number of lines to be output for box height with the 'Size' instruction, like this:

```
<  
<box 1 at b(*)+2,2 size 4 content>  
<box 2 at b(1)+1,40 size 2*20 rname>  
>
```

Box one will be four lines in length, and as is evident by the output below, a considerable amount of text has not been printed due to the lack of defined space:

```
TELEX NO 312/7
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM
PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW

Mats G. Lindquist
Mats Löfström

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,

Mr. Ron Smith

[ ]

Sirs,

I would be grateful if you could send me any information available
```

### Proportioning With Columns Only

To specify only the horizontal or left-to-right dimension, give only the size in columns for box width, leaving a blank space between 'Size' and the asterisk:

```
<
<box 1 at b(*)+2,2 size *45 content>
<box at b(*)+2,2 size 2*20 rname>
>
```

Box 1 will be forty-five columns wide, as shown by the output below:

TELEX NO 312/7

ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM

PLEASE SEND INFORMATION ABOUT THE STATUS OF

TDBS. IS IT NOW

AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE

PRICE? DOES THE

SYSTEM EXIST ON OTHER MACHINES?

RON SMITH, SPARKLER INSTITUTE

Mats G. Lindquist

Mats Löfström

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system

for VAX under VMS,

called TDBS, is under development. A first

prototype will be

at hand this summer. I will mail further

information.

Best regards,

Mats G. Lindquist

Mr. Ron Smith

Sirs,

I would be grateful if you could send me

any information available

on the free-text retrieval system 3RIP,

marketed by your company.

Thank you.

Sincerely,

George Hodge

If the width in columns is not given, the default line length will be the width of your screen or your defined printer page size, whichever is applicable.

### Box Grouping

Boxes may be assembled into box groups, collections of boxes designed to save typing, permit the creation of simpler reports and enable the sharing of material between several boxes without repeating the instructions for each box. This is done by assigning common attributes or conditional statements to more than one box, i.e. several boxes may share headers and separators. Two or more boxes may be joined to form a box group by surrounding their definitions with box group definition enclosures, the less-than [<] and greater-than [>] symbols. Most report instructions must be enclosed within a box or box group.

In this example we have added a second box, which will print the contents of the field rname (Receiver's name) directly after content. The additional < and > delimiters join the two boxes in a box group:

```
<
<<Box group functions>
<box 1 at b(*)+2,2 size *45 content>
```

```
<box at b(*)+2,2 size 2*20 rname>
>
>
```

This definition is summarized below:

Component	Explanation
<	Begin format specification
<<Box group functions>	Begin box group specification; begin and end functions that will affect all boxes in this group
<box 1	Begin box specification; define Box One
at b(*)	Position cursor on bottom line of last box written
+ 2,2	Add two linefeeds and move to column two of the new line
size *45	Make this box forty-five columns wide
content>	Print contents of field content, each line beginning in column two; end box specification
<box 2	Begin box specification; define Box Two
at b(1)	Position cursor on bottom line of Box One
+ 2,2	Add two linefeeds and move to column two of the new line
size 2*20	Make this box two lines long and twenty columns wide
rname>	Print contents of field rname; end box specification
>	End box group specification
>	End format specification

## Background Text

A text string is any collection of characters printed to screen or paper that is not native to (contained within) the database being output, and may be used in text inserts, headers, separators, and trailers. These are classified according to their dependence on field content, and may consist of text constants, data produced by output functions or the contents of format text variables.

The word element as it appears below and throughout this chapter is taken to mean those portions of a record normally accessed by a report, the field, subfield, paragraph or sentence.



Text String Type	Written As	Function
Text Insert	t	outputs where and as it appears in format
Header	h.element	labels or headlines that preface an element (field, subfield, paragraph or sentence)
Separator	s.element	separates the elements, and is not output before the first element or after the last one
Trailer	tr.element	outputs after each element has been printed

**Table 6–1 Types of background text**

None of the last three types are output when the unit the text is supposed to head, follow or separate is empty, and all three affect only the box or box group in which they occur.

TRIP recognizes certain text string reserved characters in all text string types, the meanings of which change depending on the manner in which they are used. The less-than [<] and greater-than [>] symbols in a text string signal the beginning and end of functions. The slash character [/] in a text string signifies a linefeed, and an exclamation point [!] marks a comment. To use one of these characters as literals in a text string, each must be preceded by an underscore [ \_ ].

These are outlined below:

Reserved Character	Function When Used Alone	Function When Used With Underscore
/	<CR><LF>	literal slash [/]
<	begin function	literal less-than symbol [<]
>	end function	literal greater-than symbol [>]
!	comment	literal exclamation point [!]
_	literal precursor	literal underscore [ _ ]

**Table 6–2 Text string reserved characters**

A series of spaces or tabs in a text string will be output exactly as they are written, as long as there is no linefeed in the series. If any text string in a format specification contains a series of spaces, tabs and linefeeds, one of which is a linefeed, the entire group of formatting characters will be replaced with a single space when the text is output, irrespective of their number.

A carriage return/linefeed (<CR><LF>) inside the text string of a header will not result in a carriage return/linefeed in the string that is output. To create a <CR><LF> inside a text string, you must use the slash [/], one per linefeed.

## Field-Dependent Text Strings

A field or field type-specific text string is output only if a field has content; i.e., if it is not empty. This type includes headers, separators and trailers.

### Headers

A header, abbreviated 'h', is a headline that begins a field, a subfield, a paragraph, or a sentence. The h must be followed by a period [.] and a notation for what it will head, like this:

To begin a ...	Use ...
Field	h.field or h.fieldtype or h.fieldname
Subfield	h.sub
Paragraph	h.p
Sentence	h.s

**Table 6–3 Headers**

A header applies only to the box or the box group in which it is written, and must appear before any field names in the text box definition. As with the other field-specific text string types, if the field or fields that a header is attached to are empty in a record, the header is not output.

*Note:*

*If a box or a box group contains several potential field headers, the one which is most specific to that field will have priority. For example, if a box which prints the contents of the Corr field rcomp (correspondence receiver's company name) contained instructions for h.field, h.phrase, and h.rcomp, the field-specific h.rcomp would be used.*

### Separators

A separator (abbreviation: s) segregates fields, subfields, paragraphs, or sentences, and therefore will begin printing after the first element in an output list. In the same way as for headers, the s must be followed by a full stop (period, [.]) and a notation for what it will separate:

To separate a ...	Use ...
Field	s.field or s.fieldtype or s.fieldname
Subfield	s.sub
Paragraph	s.p
Sentence	s.s

**Table 6–4 Separators**

The same rules apply to a separator as to a header when it comes to placing, contents and scope.

Using headers and separators and the example discussed previously under 'Text Box Grouping', we have altered the box definition to create columnar output for the rname and sname fields, more clearly demarcate the boundary between records and divide the text of content into cleaner paragraphs:

```
<
<box 1 at b(*)+2,2
<h.content=**/>
<s.p=/    >
content
>
<<s.sub=/>
<box 2 at b(1)+2,2 size  *40 rname>
<box 3 at t(2),40 size  *40 sname>
>
>
```

The box specification summary appears below.

Component	Explanation
<	Begin format specification
<box 1	Begin box specification; define Box One
at b(*)	Position cursor on bottom line of last box written
+ 2,2	Add two linefeeds and move to column two of the new line
<h.content	Begin header specification; define field header
=**/>	Type two asterisks, then perform one linefeed; end header specification
<s.p	Begin separator specification; define paragraph separator
= >	Perform one linefeed, then type (indent) three spaces on next line; end separator specification
content	Print contents of field content, each line beginning in column two
>	End box specification
<<s.sub=/>	Begin box group specification; separate subfields with one <CR><LF>
<box 2	Begin box specification; define Box 2
at b(1)	Position cursor on bottom line of Box One
+ 2,2	Add two linefeeds and move to column two of the new line
size *40	Make this box forty columns wide
rname>	Print contents of field rname; end box specification
<box 3	Begin box specification; define Box 3
at t(2),	Position cursor at top of Box Two
40	Move to column forty of the new line
size *40	Make this box forty columns wide
sname>	Print contents of field rname; end box specification
>	End box group specification
>	End format specification

Boxes 2 and 3 inherit the <s.sub=/> statement from their parent box group.  
Records one, three and five in Corr are considerably easier to read when output using the edited report:



not be output for that record. If a string must be output regardless of the contents of the record, you must use text inserts or field-independent text strings (short form: t) instead.

### Text Inserts

The text strings of text inserts follow the same rules as the text strings of headers and separators, in that they may contain anything that can be built into headers, separators, and trailers, and vice versa.

The only major difference between header/separator/trailer text and inserted text is that a text insert is independent of the contents of the records, and so is always output according to its position in the format specification.

The exception to this rule is that a text insert may occur in a box by itself. This is not possible with headers, separators or trailers, as these must be placed in the same box or box group as the units they head, separate or follow.

If a header marks the start of a new record, as it does in the previous example, it should be output regardless of whether the particular field that it identifies is empty or not and should be rewritten as a text insert.

If we edit the specification file from this example to make the first header a text insert, the revised code looks like this:

```
<
<box 1 at b(*)+2,2
<t=**/>
<s.p=/    >
content
>
<<s.sub=/>
<box 2 at b(1)+2,2 size *40 rname>
<box 3 at t(2),40 size *40 sname>
>
>
```

and the output will be exactly the same as that of the previous version, except that the '\*\*' string will be output even if the content field is empty.

*Note:*

*In the database Corr the content field is never empty, so a header would have worked in the unedited version.*

## Functions

### Text String Functions

There are many functions that will import information from a database, search or TRIP itself for use in text inserts, headers, separators or trailers. Here is a list of simple output functions that can be used in text strings:

Function	Output
<base>	database name
<call>	result of an external user-written function
<chr>	unprintable characters
<curdate>	current date
<dateform>	date format
<ff>	form feed
<hits>	total number of hit records in a search
<numform>	numerical date format
<occs>	number of occurrences
<pageno>	number of the printed page
<parts>	total number of record parts within the current record
<rid>	number of the record in the database
<ris>	number of the record in the search
<rname>	record name
<subrid>	number of the record part being output within the record
<substring>	substring isolated from a given element; also used to produce right-justified output
<timeform>	time format
<weight>	rank of a record after a fuzzy search, or the number of hits after a non-fuzzy search

**Table 6–6 Text string functions**

*Note:*

*For a detailed presentation of each function, refer to the ‘Output Format Reference Guide’ at the end of this chapter.*

Building on the previous example, we have chosen to output the field scomp (Sender’s company) rather than rname in Box Two, and added the <RID> function to the text insert that prefaces each printed record. This will output the database record number of the record being printed:

```
<
<box 1 at b(*)+2,2
<t=** Record No. <rid> **/>
<s.p=/    >
content
>
```

```
<box 2 at b(1)+2,2 size *40 scomp>
<box 3 at t(2),40 size *40 sname>
>
```

Here is the output for record numbers one, three and five using the altered format:

```
** Record No. 1 **

Dear Mr. Smith,

Thank you for your telex. The status of TDBS is as follows:
the central modules of the system are completed and work on
the user interface is underway. We will exhibit the system
in Stockholm in November, and at that time will have some
new material about the system, which I will send you.

The first version of the system is, as you know implemented
on a VAX in Pascal. He will make the system portable to other
machines, e. g. IBM, in the near future.

Hoping that you can hold out a little bit longer, I remain

Yours sincerely,
Mats G. Lindquist
Marketing Manager

Paralog AB                                Mats G. Lindquist

** Record No. 3 **

TELEX NO 312/7
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM
PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IS IT NOW
AVAILABLE ON VAX11/780? WHAT IS THE PURCHASE PRICE? DOES THE
SYSTEM EXIST ON OTHER MACHINES?
RON SMITH, SPARKLER INSTITUTE

The Sparkler Institute                      Mr. Ron Smith

** Record No. 5 **

Dear Mr. Smith,

3RIP runs only on DEC10 and DEC20. A system for VAX under VMS,
called TDBS, is under development. A first prototype will be
at hand this summer. I will mail further information.

Best regards,

Mats G. Lindquist

PARALOG AB                                Mats G. Lindquist
```

### Field-Dependent Text Functions

The 't=' functions listed previously may be used either as text or conditional (field-dependent) functions. In addition, there are six conditional or <For> loop functions whose use is dependent on field content, as listed in the table below:



<For> Loop Function	Returns:
<fieldname>	field name, in capital letters
<fieldtype>	data type
<fieldnumber>	number of the field within the record
<subfieldnumber>	number of the subfield within the field
<paragraphnumber>	number of the paragraph within the field
<sentencenumber>	number of the sentence within the paragraph

**Table 6–7 Field type-dependent functions**



*Note:*

*The above functions can only be used in headers, trailers and separators, not in text inserts.*

The lengths of output functions of all types may be made consistent, as shown in this example:

```
<h.field=<fieldname(10)>>
```

This format writes the field name left-aligned, followed by as many spaces as are needed to make the item ten characters long, and trims it if it is longer than ten characters.

### Sample Output Format

We can construct a report that can be used with any TRIP database by using many of these functions. To make the format independent of particular fields, we will use data type names rather than field names to output field contents, as seen below:

```
<
  <box at b(*)+4,2
  <t>//Record No. <rid>>
  <h.field=//<fieldname> (type <fieldtype>):/>
  <h.s=//<paragraphnumber>.<sentencenumber>: >
  <s.field=//><s.p=//>
  <h.sub= <subfieldnumber>: ><s.sub=, >
  text phrase number date time
>
>
```

This format prints all non-empty fields of a record, each headed by its name and field type. The fields are output in the order given by the list of data types (Text fields first, then PHrase, etc.), and in ordinal field number order within data type.

Here, record number one in Corr is shown in the new format:

Record No. 1

CONTENT (type TEXT):

1.1:

Dear Mr.

1.2: Smith,

Thank you for your telex.

1.3: The status of TDBS is as follows:

the central modules of the system are completed and work on  
the user interface is underway.

1.4: We will exhibit the system

in Stockholm in November, and at that time will have some  
new material about the system, which I will send you.

2.1: The first version of the system is, as you know implemented  
on a VAX in Pascal.

2.2: He will make the system portable to other  
machines, e. g.

2.3: IBM, in the near future.

3.1: Hoping that you can hold out a little bit longer, I remain

Yours sincerely,

Mats G.

3.2: Lindquist

Marketing Manager

RNAME (type PHrase):

1: Mr. Ron Smith

RCOMP (type PHrase):

1: The Sparkler Institute

RADDR (type PHrase):

1: 16 Sparkling Road, 2: Sparkletown

RCOUNTRY (type PHrase):

1: USA

SNAME (type PHrase):

1: Mats G. Lindquist

SCOMP (type PHrase):

1: Paralog AB

SADDR (type PHrase):

1: Box 2284, 2: 103 17 STOCKHOLM

SCOUNTRY (type PHrase):

1: Sverige

MODIFIED (type PHrase):

1: SYSTEM, 2: SYSTEM, 3: VIOLA

DAY (type DATE):

1: 15-Jun-84

MODDATE (type DATE):

1: 17-Aug-93, 2: 17-Aug-93, 3: 18-Jun-93

```
MODTIME (type Time):  
1: 10:50:44, 2: 10:50:18, 3: 11:51:42
```

Since there is only one box, all fields share the headers and separators, which in the specification are placed before the single text insert and the data type names that signal the output of field contents. Every sentence (Text fields only) is headed by its paragraph and sentence numbers, and every subfield (all other data types) is headed by its subfield number. The fields as well as the paragraphs are separated by a single line feed, and the subfields by a comma and a space.



## Box Functions

These standalone functions affect either the manner in which data is output in boxes and/or box groups or the information that is output by the entire box or box group.

Function	Output
<at_end>	causes output generated by its host box to be printed only once, after the last record
<case>	outputs values that are dependent on the value of an element
<if-changed>	causes output only if the value to be output has changed since the record was last printed
<if-empty>	causes output only if the appropriate element is empty
<if-nonempty>	causes output only if the appropriate element is not empty
<if-unchanged>	causes output only if the value to be output has not changed since the record was last printed
<indent>	indents all lines except the first by n number of characters
<link>	causes output from a secondary database depending on the value of a given field in the current database
<noorig>	suppresses the default output layout
<once>	causes output generated by its host box to be printed only once, before the first record
<orig>	returns text field output to its original layout
<trace>	gives search history preceding the search result being output

**Table 6–8 Box and box group functions**

## Format Functions

These functions have a global effect on the format, and all except <noff> and <nolf> work on the record level.

Function	Effect
<text variables>	defined on a per-record basis as a placeholder for later record output
<call>	allows pre-output modification of record content (in memory)
<debit>	sets minimum and maximum unit cost for any output created with a format
<noff>	suppresses automatic FF in printed output
<nolf>	Suppresses automatic line planning
<sortfields>	defines fields to be used as sort keys

**Table 6–9 Format functions**

## <For> Loops

### General Structure

<For> loops allow the user to define a variable to run using either a range or a list of values where:

- the default is one up to the maximum number of parts or subfields being output by the box(es) in the loop
- the maximum value is dictated by the use to which the variable is put within the boxes in the loop.

*Note:*

*Be careful to use a variable ONLY for subfields OR parts, NOT both.*

<For> loops use this general syntax:

```
<For <variable>
... boxes ...
>
```

The <For> construct enables looping through the record components in records with record parts, and is also useful for outputting field elements. <For> is analogous to a box group in that headers, separators, trailers and box group functions can be used outside of boxes within a <For> loop. A control variable associated with <For> may either cycle through all the components of a record, or only those located via search.

Function	Effect
<loop variables>	represented by any alpha character from A to Z and used to denote the current loop index
<append>	prevents positioning of the box to be executed from the second through the n <sup>th</sup> loop
<hitlist>	causes the list of values for the FOR loop to be restricted to those part records located by the search being output

**Table 6–10 FOR loop functions**



## Examples

The first example is taken from the head/part demonstration database Carroll, the structure of which is outlined in the figures below.

### Example 1:

In Carroll, the field person is a head field and speaker and txt are part fields.

This format outputs the chapter and book names, persons acting in each chapter, speakers in the text, verse and text of each part record:

```
<
<box 1 at b(*)+1,1
<t=Chapter >
chaptnr
<t=, '>
chapter
<t=' from ">
book
<t=">
>
<box 2 at b(*)+2,3
<s.sub=, />
<h.field=Persons in Chapter : >
<indent(21)>
person
>
<for <a>
<s.s= ><s.p=/ >
<box 3 at b(*)+2,3
<s.sub=, />
<h.field=Speakers on Page : >
<indent(21)>
.a.speaker
>
<<s.s= ><s.p=/ >
<box 4 at b(*)+2,3 .a.txt>
<box 5 at b(*)+2,3 .a.verse>
<box 6 at b(*)+2,3 .a.txt2>
>
>
>
```



The first and part of the second subfield of Record 1 are shown below in the new format:

```
Chapter 1, 'Down the Rabbit-hole' from "Alice's Adventures in Wonderland"

Persons in Chapter :      Alice's sister
                        White Rabbit
                        Alice's family
                        Dinah

Speakers on Page : White Rabbit

Alice was beginning to get very tired of sitting by her sister on the bank,
and of having nothing to do: once or twice she had peeped into the book her
sister was reading, but it had no pictures or conversations in it, "and what
is the use of a book," thought Alice, "without pictures or conversations?"
So she was considering, in her own mind (as well as she could, for the hot
day made her feel very sleepy and stupid), whether the pleasure of making a
daisy-chain would be worth the trouble of getting up and picking the daisies,
when suddenly a white rabbit with pink eyes ran close by her.

Speakers on Page : White Rabbit

There was nothing so VERY remarkable in that: nor did Alice think it so VERY ...
```

### Example 2:

The fictional database in the next example contains results from the Olympic games. The head record contains the event location and dates, and there is one record part for each event and its winners.

The structure of hypothetical database Olympic\_Games is shown below:

Field name	Type	Part	Comment
Place	PHrase	N	Location of games
When	DAte	N	Date of event
Event	PHrase	Y	Name of event
Medal	PHrase	Y	Medal awarded
Winner	PHrase	Y	Winner's name
Nation	PHrase	Y	Winner's nation
Result	PHrase	Y	Winner's score

**Table 6–11 Structure of Olympic\_Games**

This format prints the name of each Olympic event, the medal won, the winning score and the name and home country of each medal winner.

```
<
<box 1 at b(*)+1,1
<t=Olympic Games, <dateform(when.1,15,//)> at >
place
>
<for <x>
```

```
<box 2 at b(*)+2,3
<t=In the >
.x.event
>
<for <y>
<box 3 at b(*)+1,3> .x.medal.y>
<box 4 at t(3),10 .x.result.y>
<box 5 at t(3),20 .x.winner.y
<box 6 at t(3),40 <t=from >.x.nation.y>
>
>
>
```

**Note:**

*Only one index variable may be used for all part loops in a format (i.e. constructs of the type .x.event).*

The record for the summer games of 1976 has been output using this format as shown below:

Olympic Games, 15/Jul/1984 at Montreal, Canada

In the Men's Basketball

Gold			from USA
Silver			from Yugoslavia
Bronze			from Bulgaria

In the Women's Basketball

Gold			from USSR
Silver			from USA
Bronze			from Bulgaria

In the Men's Archery

Gold	2571.2	D. Pace	from USA
Silver	2502.3	H. Michinga	from Japan
Bronze	2495	G. Ferrari	from Italy

In the Women's Archery

Gold	2499.2	L. Ryon	from USA
Silver	2460.3	V. Kovpan	from USSR
Bronze	2407	Z. Rustamova	from USSR

### Example 3:

This example prints the location and date of the games, and the medal and winning country for each event.

```
<
  <box 1 at b(*)+1,1
  <t=Olympic Games at >
  place.1
  <t=, >
  when.1
>
<for <x>
  <box 2 at b(*)+2,3
  <t=In the >
  .x.event
  <t=/>
>
  <for <y>
  <box 3 at b(*)+1,5>
  <append>
  <s.field= - >
  <s.sub=_/_/ >
  .x.medal.y>
  .x.winner.y
  .x.nation.y
  .x.result.y
  >
  >
  >
  >
```

Sample output for the 1976 summer games is given below:

```
Olympic Games at Montreal, Canada,          1-Jun-1976

In the Men's Basketball

    Gold - USA // Silver - Yugoslavia // Bronze - Bulgaria

In the Women's Basketball

    Gold - USSR // Silver - USA // Bronze - Bulgaria

In the Men's Archery

    Gold - D. Pace - USA - 2571.2 // Silver - H. Michinga - Japan - 2502.3 //

    Bronze - G. Ferrari - Italy - 2495

In the Women's Archery

    Gold - L. Ryon - USA - 2499.2 // Silver - V. Kovpan - USSR - 2460.3 //

    Bronze - Z. Rustamova - USSR - 2407
```

## Page Control

### Page Level Boxes

Up until now we have been considering output only as a continuous stream, however TRIP makes provision for the arrangement of paged output, for example, printing a trailer at the bottom of a hard-copy page.

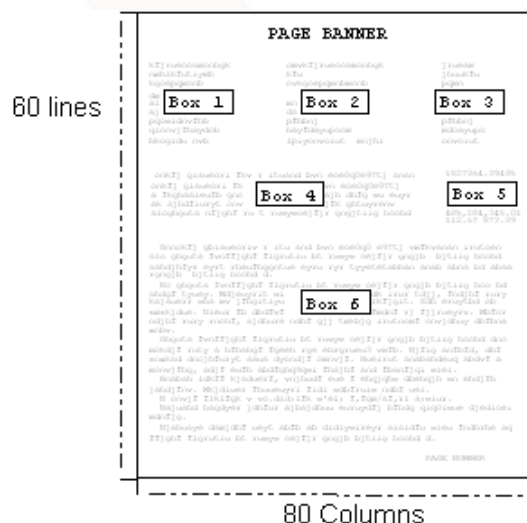


Figure 6-7 Paged output

Headers and footers can be defined for an output page using `header_box` and `trailer_box` constructs. Page headers and trailers are not output if the format is used when FOCUS is active.

### Header\_Box

A `header_box` is output at the top of each page. If the box contains any record-specific information, the data of the top record on the page is used. For example, this code,

```
<header_box size 1*11 <t=Page Header>>
```

results in the text 'Page Header' being printed at the top of each output page.

*Note:*

*TRIP does not recognize or act on positional information (i.e. at 1,1) used in header or trailer boxes.*

This code from database Carroll,

```
<header_box size 1*80 <t=Book: > book>
```

results in the text string 'Book:' and the contents of the field book being written on the first line of each output page.

*Note:*

*In this instance, when the value of book changes and the record that institutes the change begins printing on any line on the page after the first one, the previous value will be output. To avoid this, use <if-changed> and <FF> statements in the box definition.*

### Trailer\_Box

A trailer\_box is output at the bottom of each page. If the trailer\_box contains any record-specific information, the data of the bottom record on the page is used. For example,

```
<trailer_box size 3*40 <t=/End of Page/Page  
Number <pageno>>>
```

reserves three lines for the trailer\_box at the bottom of each page. The first line will be empty, the second one will contain the text 'End of Page', and the third line the text 'Page Number' and the page number.

### Page Size

The default page size for Print or hard-copy formats is the normal screen size, twenty-four lines by eighty columns. To change to something more suitable for a printer, include a size specification at the start of the format, as in the following example:

```
<page size 72*60  
<box at b(*)+2,2  
<s.p=/ >  
<t=** Record no. <RID> **/>  
content  
>  
<box  
<h.field=/ >  
scomp  
>  
>
```

This page size will be disregarded when the format is used with a Show order, which uses the default size. The product of the number of lines and number of columns must not exceed 8192 (8K).

*Note:*

*If you are using header\_box or trailer\_box constructs, you must define a page size, or these will be output using the default page size specification. This will cause two or three headers and/or trailers to be printed on every page.*

### Columnar Output

You may have the output printed in columns by giving the page size as '1\*c/k', where 1 is the number of lines on the page, c is the number of characters in the line, and k is the number of columns on the page. Here is an example of a format for printing the address labels from Corr in two-character wide columns:

```
<page size 60*80/2
<box 1 at b(*)+1,1 size 8*40
<s.sub=/>
rname
rcomp
raddr
rcountry
>
>
```

### Output Formats for Database Clusters

When a user starts searching in a database cluster the default formats of the individual databases are used. If a DEfine Format order is given, calling up a new format, the system will look for this format for all the databases open, and if it does not exist for one of them, the previous format will remain in use for that database.

If the page size has been defined for one report in a cluster database, the same page size must be defined for all databases in the cluster. An alternative to this is to specify the page size in the print control file.

When the formats for the individual databases are uncoordinated, the user can easily become confused, therefore we recommend that you keep this in mind when constructing and naming formats for databases that are meant to be searched and shown together.

## Related CCL Commands

To view a list of all the existing reports for a database, give the order:

```
SHOW format [BASE=dbname]
```

The Show window looks like this:

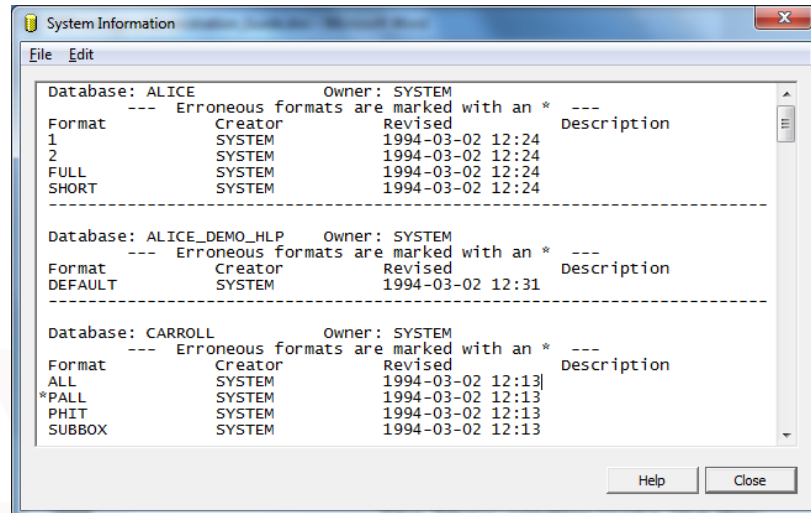


Figure 6–8 The Show Format window

## Output Format Reference Guide

Each item in this section has been provided with a general description, a scope of action, proper syntax, known side effects and examples.

### <APPEND>

#### Description

If placed in a box within a <for> loop, this function directs the reporter to ignore that box's positioning clause for the second through nth iteration of the loop. Positioning of elements is thus controlled by header/separator/trailer usage rather than by box layout.

#### Scope

Layout box or box group function

#### Syntax

```
<append>
```

#### Side effects

- All fields to be output by the <for> loop being affected by the <append> function must be contained within a single box.
- Trailers can be used in for loops containing <append>

#### Examples

##### Example 1:

```
<for <y>      ! Subfield loop
<box 1 at b(*)+3,1
<append>
<s.field= - >
<s.sub= \\ >
field1.y
field2.y
>
>
```

In this example, the positioning clause for box 1 (b(\*)+3,1) would only be obeyed for the first subfield output by the 'y' <for> loop. In all other cases, the field separator would be used to position each field within the box, while the subfield separator would be used to position each new instance of the loop.

Considering a record containing field values of :

Field 1 : a, b

Field 2 : d, e, f



we would see the output :

```
[      ]  
[      ]  
a - d \\ b - e \\ f
```

where the empty brackets [] indicate empty lines inserted by TRIP.



## <AT\_BEGIN>

### Description

If placed in a layout box, that box will only appear once in the resulting output, during the display of the first record.

See also <ONCE>, which is similar but applies to each database in the output.

### Scope

Layout box or box group function

### Syntax

```
<at_begin>
```

### Side effects

None

### Examples

#### Example 1:

To output a cover page for printed output:

```
<
  <box at b(*)+1,1
  <at_begin>
  <t=This print was produced on <curdate>.>
>
...
>
```

## <AT\_END>

### Description

If placed in a layout box, that box will only appear once in the resulting output, during the display of the last record.

### Scope

Layout box or box group function

### Syntax

```
<at_end>
```

### Side effects

None

### Examples

#### Example 1:

To output a summary page for printed output:

```
<
  <box at b(*)+1,1
  <at_end>
  <t=This print was produced on <curdate>.>
>
...
>
```

## <BASE>

### Description

Returns a string containing the name of the physical database from which the record being output originates.

### Scope

Text string function

### Syntax

`<base>` or `<base(n)>`

where n is the width of the string to be returned. If the actual string length is less than that specified by n, the string is padded with blank space characters (ASCII 32).

### Side effects

None

### Examples

#### Example 1:

```
<box at b(*)+1,1  
<t=This record is from database <base>.>  
>
```

which would result in output such as:

```
This record is from database ALICE.
```

## <CALL>-Format

### Description

Directs the report formatter to place a call to a user-written subroutine before attempting to format the record about to be output.

### Scope

Format function

### Syntax

```
<call (ase_name, literal)>
```

where ase\_name is the name of the user-written subroutine to be called, and literal is a quoted string such as 'abc', which is passed without modification to the user-written subroutine.

### Side effects

The record which is about to be formatted for output is contained in the system record control structure, to which a handle may be retrieved by calling TdbCurrentItem(). The contents of this structure may be modified in memory (this is the intended function of the format level <call> function), but no attempt should be made to write the modified record back to the originating database.

Any attempt to alter the number of parts in a record or the number of paragraphs in a TExt field will produce no effect in the output, as these values are computed upon reading the record and cannot be changed during output.

For more information, please consult the Appendix in this manual.

### Examples

#### Example 1:

```
<  
    <call (my_ase, "") >  
    <box at b(*)+1,1 field1 >  
>
```

Assuming that the user-written subroutine 'my\_ase' alters the value of the field field1 in some way, the formatter will output the newly altered value rather than the original value stored in the database.

This function can be particularly useful when applied to the filling of blank or dummy fields. Such fields exist in the database design solely for use during report formatting, for example, to hold a computed value such as a number field column total:

my\_ase is:

```
total = 0
for each subfield in field values do
    total = total + current subfield of VALUES
done
put total into field column_total
```

which can be used in a report such as:

```
<
<call( my_ase, "" )>
<box at b(*)+1,1
<s.sub=/>
values
>
<box at b(*)+1,1
<h.field=--/>
column_total
>
>
```

resulting in (for example):

```
10
20
--
30
```

## <CALL>–Text String

### Description

Directs the report formatter to place a call to a user-written subroutine specified, before attempting to format the text string for output. The user-written subroutine is thus responsible for returning the text insert to be output.

For more information, please consult the Appendix in this manual.

### Scope

Text string function

### Syntax

```
<call (ase_name, field_element[, delay])>
```

or

```
<call (ase_name, literal[, delay])>
```

where `ase_name` is the name of the user-written subroutine to be called, `field_element` is an element of a field, such as a subfield or sentence, which is to be passed to the user-written subroutine for processing, `literal` is a literal quoted string such as 'abc', which is to be passed to the user-written subroutine for processing, and `delay` is the point in the output at which the user-written subroutine is to be called:

0 (or omitted)	call immediately
1	call at end of page
2	call when user presses the graphic key (in TRIPclassic only; <Gold><G>).

### Side effects

None

### Examples

#### Example 1:

```
<
<box at b(*)+1,1
<t=The result of MY_ASE is <call (my_ase,
"", 0)>.>
>
>
```

which, assuming that 'my\_ase' returns a string such as 'ZABULON', would give the output:

```
The result of MY_ASE is ZABULON
```

For information concerning the method of argument passing and string return values, please consult the Appendix in this guide.

A simple example of the use of this type of ASE call is the computing of an imperial or non-metric measurement from a stored metric measurement, a conversion from Celsius to Fahrenheit, etc.

## <CASE>

### Description

The <case> function results in a text string, which is dependent upon the contents of a particular field element within the record being output. This text string can be used either directly as output or to load a value into a text variable.

For instance, if the record contains a simple Yes/No field in 'Y' and 'N' form, but users would rather see 'Yes' and 'No', a case function provides a simple method of achieving this. Thus:

```
if the record contains 'Y' then
  output 'YES'
else if the record contains 'N' then
  output 'NO'
```

### Scope

Layout box or box group function

### Syntax

```
<case (field_element, list)>
```

where field\_element is the component of the record, typically a field plus subfield combination, that is used as the 'selector' of the <case>, and list is a comma-separated list of selector/value pairs that defines the output for a given selector.

The format of the list is:

```
selector1 : value1, selector2 : value2, ..., [ selectorn ] : valuen
```

where the omission of the last selector (selectorn) signifies that when the field\_element value does not match any given selector, the last value (valuen) will be output.

*Note:*

*both selectors and values should be surrounded by single quotation marks if they are to be interpreted as literal values. If not, they are interpreted as field names.*

### Side effects

None



## Examples

### Example 1:

This example uses <case> instead of a field name for direct output:

```
<box at b(*)+2,10
  <case(yesno.1,
    'Y': 'Yes',
    'N': 'No',
    : 'Maybe')>
>
```

which, depending on the value of the first subfield of the field yesno, will output either 'Yes', 'No', or 'Maybe'.

### Example 2:

This example uses <case> to assign a value to a text variable, where the value to be assigned originates in a different subfield of the current field than that which is being used as the selector:

```
<
  <l=<case(selector.1,
    '0': selector.2,
    '1': selector.3,
    '2': selector.4,
    : selector.5)>>
  <box at b(*)+1,1 <t=<l>>>
>
```

If the field selector had values as shown below (where the comma delimits subfields):

```
Record 1 : 0, Hello
Record 2 : 1, xxx, my
Record 3 : 2, xxx, xxx, name is
Record 4 : 3, xxx, xxx, xxx, Jim
```

the output would be:

```
Hello
my
name is
Jim
```

## <CHARSET>

### Description

Inserts the standardized name of the current session's character set. This is "UTF-8" for Unicode UTF-8 sessions and "windows-1252" for sessions using LA1.

### Scope

Text string function

### Syntax

```
<charset>
```

### Side effects

None

### Examples

#### Example 1:

An output format that generates an XML document can include this filter to assign value to the encoding attribute of the XML declaration statement:

```
<
  <nolf>
  <entitify>
  <box at b(*)+1,1
    <once>
      <t=_<?xml version="1.0"
encoding=_ "< charset>" standalone=_ "no_"_?_>>
      <t=/_<documentroot_>>
    >
```

## <CHR>

### Description

In order to use unprintable characters in a report (such as the escape character <Esc> or ASCII 27), you must use the <chr> function, which allows any number of ASCII character values to be inserted into the output stream.

### Scope

Text string function

### Syntax

```
<chr (list)>
```

where list is a comma-separated list of ASCII decimal values that signifies the unprintable characters to be output.

### Side effects

If the format using <chr> is to be used for Show output, you must ensure that the characters being used will not adversely affect the state of the user's output device, e.g. his or her terminal. Inserting an XOFF character within the output, for example, would effectively lock the terminal from accepting input.

A <chr> character does take up a character position, thus influencing the number of characters on a line.

### Examples

#### Example 1:

The DEC printer escape sequence for enabling the 'Bold' character attribute uses the ASCII escape value 27 (<Esc>[1m):

```
<t=<chr (27)>[1m>
```

## <CLASS>

### Description

Retrieves the name of the class that has been associated with the currently active record (if any). For more details on classification see "Appendix B – Classification Schemes" on page 269 of this manual and the CCL Command reference, "Display CLASS()" command.

### Scope

Text string function

### Syntax

```
<class>
```

### Side effects

None

### Examples

#### Example 1:

To list the category for the current record:

```
<t=CATEGORY: <class>>
```

## <CURDATE>

### Description

Returns a string containing the current date in the format defined by the user's profile.

### Scope

Text string function

### Syntax

```
<curdate>
```

### Side effects

This function may also be used in the date or time formatting functions <dateform> and <timeform> to produce the current date or time in a particular format. See the descriptions of <dateform> and <timeform> for more information.

### Examples

#### Example 1:

```
<box at b(*)+1,1  
<t=The date is <dateform(<curdate>,15,--)>/>  
<t=The time is <timeform(<curdate>,1)>>  
>
```

which could result in the output:

```
The date is 7-Nov-93  
The time is 19:30:57
```

## <DATEFORM>

### Description

The <dateform> function is used to modify the output form of a date value, specified either by a DATE field or by using the <curdate> function to return the current system date. The <dateform> function supports seventeen different date formats:

Numbered Date Form	Sample Date
1	1993-05-01
2	1993-5-1
3	93-5-1
4	1993-May-1
5	93-May-1
6	05-01-1993
7	5-1-1993
8	5-1-93
9	May-1-1993
10	May-1-93
11	01-05-1993
12	1-5-1993
13	1-5-93
14	1-May-1993
15	1-May-93
16	19930501
17	930501

**Table 6–12 Date formats**

### Scope

Text string function

### Syntax

```
<dateform(value, format, separators)>
```

or

```
<dateform(value, format)>
```

or

```
<dateform(value, 0, separators)>
```

where value is the DATE value to be output by the function. This value can be specified as a DATE field, with or without subfield, or as the current system date by using the <curdate> function.

Format is an integer value between 1 and 17 (as outlined in the previous table), where each value specifies a unique date format.

Separators are two literal characters which specify the separators to be used between the date elements, i.e. the year and the month, and the month and the day. If none are specified, the date separators are taken from the user's profile. Valid values for each separator character are slash [/], hyphen [-], period [.] , and colon [:]. Space is also possible to use as a separator, but requires that the value to be set a caret [^] instead of an actual space.

0 (zero), which, when used as the format argument, specifies that the function should use the format specified by the user's profile. This is intended to allow the format to output the user's preferred date type, but with the format designer's preferred date separators.

### Side effects

None

### Examples

The <dateform> function can be used to output a date value in a specific form, output the current date in a specific form or modify the date separators in the user's preferred date form:

#### Example 1:

```
<t=<dateform(my_date.2, 15, //)>>
```

Format the second subfield of the field my\_date using date format 15, and separators [/]. Assuming that the value of my\_date.2 is 2nd October, 1993, the output would be

```
2/Oct/93
```

#### Example 2:

```
<t=<dateform(<curdate>,15,-/)>>
```

formats the current system date using date format 15 and dash [-] and slash [/] separators:

```
25-Nov/93
```

#### Example 3:

```
<t=<dateform(<curdate>,0,:)>>
```

formats the current system date using the user's preferred date format, but modifies the date separators to the double colon [::]. Assuming that the user has a preferred format of 5:

```
93:Nov:25
```

#### Example 4:

```
<t=<dateform(my_date,15)>>
```

formats the first subfield of the field my\_date using the date format 15, taking the date separators from the user's preferred settings (as specified in his or her user profile). Assuming the separators to be double dashes [--] and my\_date to hold 1st October, 1993:

```
1-Oct-93
```

## <DEBIT>

### Description

When a database has an attached accounting function (as specified by a cost for any of the fields of that database), the output cost of any record within that database may be controlled by using the <debit> function. This function allows the specification of a minimum and maximum value for record output, for instance, to specify that each record will incur at least a unit cost of 5, but never more than 15, regardless of the individual field costs specified in the database design.

### Scope

Format function

### Syntax

```
<debit (minimum, maximum)>
```

where minimum specifies the minimum unit cost and maximum defines the maximum unit cost that will be incurred for each record output by using this report.

### Side effects

None

### Examples

#### Example 1:

```
<  
  <debit (5, 15)>  
  <box at b(*)+1,1 field1>  
>
```

Sets the minimum cost of record output to 5, and the maximum to 15.



## <ENTITIFY>

### Description

Converts characters having a special meaning in XML and HTML into their corresponding character entities before being emitted to the report. The characters that will be converted are:

Character	Entity
&	&amp;
"	&quot;
'	&apos;
<	&lt;
>	&gt;

Place this filter at the very beginning of the output format specification.

### Scope

Format function

### Syntax

```
<ENTITIFY>
```

### Side effects

None

### Examples

#### Example 1:

```
<  
  <nolf>  
  <entitify>  
  ...
```

## <FF>

### Description

Either fills the output page with blank space (during Show), or forces a hard page throw (during PPrint) by outputting a form feed character (ASCII 12, <FF>). This function can be useful in many cases, from forcing a new page when a value used in a page heading changes to protecting a user from involuntary viewing of chargeable material.

### Scope

Text string function

### Syntax

<FF>

### Side effects

None

### Examples

#### Example 1:

Using the <ff> function to throw a page whenever a specified field value changes:

```
<
  <header_box size 3*80
  <t=/----- > author <t=-----/>
>
  <box at b(*)+1,1
  <if-changed(author)>
  <t=<FF>>
>
  ...
>
```

which will begin a new page whenever the author field changes value. For more detail on the use of <if-changed()>, consult its reference section later in this chapter.

**Example 2:**

Using <FF> to protect unwilling output of chargeable material:

```
<
<box 1 at b(*)+1,1
<t=Author's name: >
author
>
<box 2 at b(*)+1,1
<t=Document title: >
title
>
<box 3 at b(*)+2,1
<t=The viewing of the content of this >
<t=document incurs a cost.\ >
<t=In order to avoid this cost, use the >
<t=command "Next".\ >
<t=To view the content of the document, >
<t=use the command "More"\".>
<t=<FF>>
>
<box 4 at b(*)+1,1
content
<t=<FF>>
>
>
```

which might result in output as follows:

```
Author's name: Slartibartfast
Document title: "Earth" - A Design Experiment
The viewing of the content of this document incurs
a cost. In order to avoid this cost, use the
command "Next". To view the content of the
document, use the command "More".
```

## <FOR> Loops

### Description

The purpose of a <for ...> loop is to direct the report formatter to loop over the individual elements of the entities included within the loop, either subfields or part records. This allows the report designer some control over the order and placement of the subfields of a field, or over the part records from a meta-record without having to know in advance how many subfields or part records there are likely to be in a given record.

Using the standard entity addressing nomenclature of the report formatter, any individual subfield, paragraph, sentence or part record can be output using:

Element	Example	Explanation
subfield	field_name.4	fourth subfield of any non-TEText type field, such as PHrase, NUmber, etc.
paragraph	field_name.2	second paragraph of any TExt type field
sentence	field_name.2.4	fourth sentence of 2nd paragraph of any TExt type field
part	.2.any_field	second part record of the current meta-record

In most cases this type of addressing will be inadequate, as the number of the subfields or parts, to be output will not be known at the time of report design. The <for> loop allows the designer to treat the entire conglomerate of elements as a single case, rather than having to write output code for each subfield or part record.

Thus, when looping over part records in a meta-record, the part records to be output are addressed using the construct:

```
.x.field_name
```

while an individual subfield within a field is addressed using a suffix construct:

```
field_name.x
```

where x in the above nomenclatures is the name given to the index of the <for> loop, and has bounds from 1 to the maximum number of subfields or parts which make up the entity being output.

For instance, if the meta-record being output in the first example had three part records, then x would have bounds of 1 to 3. If the field being output by the second example had 28 subfields (any of which may be blank), the bounds of x would be 1 to 28.

The <for> loop construct can be used to output fields in a tuple, fields from a part record, fields in a tuple from fields in a part record, etc.

### Scope

Conditional function

## Syntax

```
<for <idx>
    ... boxes containing element output code ...
>
```

where <for> loop name idx is a single alphabetic character between 'a' and 'z', and represents the current index of the loop.

## Side effects

The <for ...> construct doubles as a box group, so any functions which have group scope will also work correctly within a <for> loop. You can also nest a box group within a <for> loop.

You should not use the same index name more than once in the same format, as the results are unpredictable.

## Examples

### Example 1:

A typical use of a <for> loop is to control the output of fields, which have been declared as belonging to a tuple on a data entry form. The output of such fields must be controlled by a loop in order to stop the report formatter from suppressing the output of blank subfields, which would destroy the integrity of the tuple.

```
<for <a>
    <box 1 at b(*)+1,1 given_name.a>
    <box at t(1),30 surname.a>
    <box at t(1),60 middle_init.a>
>
```

This example uses the loop index 'a' to control the output of the subfields from the fields given\_name, surname and middle\_init. This could result in output such as:

Value of a:	Given_Name	Surname	Middle_Init
1	Gwyn	Fisher	
2	Al	Burgasser	J
3	Slarti	Bartfast	X

As you can see, the position of the middle initial for 'Al J. Burgasser' is maintained even though subfield 1 for field middle\_init is empty. If this same information had been output using:

```
<
<
< s.sub= />
< box 1 at b(*)+1,1 given_name>
< box at t(1),30 surname>
< box at t(1),60 middle_init>
>
>
```

the results would be:

Given_Name	Surname	Middle_Init
Gwyn	Fisher	J
Al	Burgasser	X
Slarti	Bartfast	

which, as you can see, does not respect the blank subfield in middle\_init.

## <HITLIST>

### Description

When used as a modifier to a <for> loop variable, the <hitlist> function directs the formatter to only output those part records which have been hit by the search set being output. If the search being output only hit the head records, or was a record search such as:

```
BASe xyzyy
```

or

```
Find R=FRom 1
```

or

```
Find
```

no part records will be output.

This function has no effect on the output of subfields in a tuple being controlled by a <for> loop.

### Scope

For loop function

### Syntax

```
<for <x:<hitlist>> ... >
```

where x is the loop variable being used to control output of part records.

### Side effects

None

### Examples

#### Example 1:

The following format will only output the speaker field from those parts which are hit by a search in the TRIP demonstration database Carroll. None of the head fields will be output, but each record is marked by a banner to signify the start of said record:

```
<
<box at b(*)+1,1
<t=Record <rid> from database CARROLL>
>
<for <a:<hitlist>>
<box at b(*)+1,1
<t=Part <subrid> :->
>
<box at b(*)+1,3
<s.sub=/>
.a.speaker
>
>
```

```
<box at b(*)+1,1 <t=/> >  
>
```

A search sequence such as:

```
S=1 <24> BASE CARROLL  
S=2 <3> Find hatter
```

might result in:

```
Record 6 from database CARROLL  
Part 17 :-  
Cheshire Cat  
Part 22 :-  
Record 7 from database CARROLL  
Part 1 :-  
March Hare  
Mad Hatter  
Part 3 :-  
March Hare  
Mad Hatter  
Part 4 :-  
March Hare  
Mad Hatter  
Dormouse
```

etc.

Alternatively, the search sequence

```
S=1 <24> BASE CARROLL  
S=2 <3> Find PERSON=hatter
```

would result in the following output:

```
Record 6 from database CARROLL  
Record 7 from database CARROLL  
Record 11 from database CARROLL
```

as person is a head field and thus no part records would be hit by the search.



## <HITS>

### Description

Returns a text string containing the number of records hit by the search being output.

### Scope

Text string function

### Syntax

`<hits>`

or

`<hits(n)>`

where n is the maximum number of characters that the string returned should contain. If the string to be returned is shorter than n, the string is padded with blank space (ASCII 32) characters.

### Side effects

None

### Examples

#### Example 1:

```
<
  <box at b(*)+1,1
  <t=The search being output consists of >
  <t=<hits> records.>
>
...
>
```

which could result in:

```
S=n   <10> Find XYZZY
```

```
The search being output consists of 10 records.
```

## <IF-CHANGED>

### Description

Controls the output of a box or box group depending on whether the contents of a field, or a number of fields, have changed since the last record output. Optionally, the function can fail if this is the first record being output.

*Note:*

*This function cannot be used to test whether a given subfield has changed; thus, all tests are performed on the first subfield or sentence of the field(s) in question.*

### Scope

Layout box or box group function

### Syntax

```
<if-changed(field_name[, field_name...][, flags])>
```

where `field_name` is the name of a field or a list of comma-separated fields whose contents are to be tested against their values during the last record output, and `flags` is an optional integer which can take the following values:

- 1 Instead of applying an AND operation between the fields given, the report formatter will use an OR. That is, if any of the fields listed have changed, the function succeeds. If this is the first record being output, the function will not succeed.
- 2 If this is the first record being output, the function will not succeed, i.e. the box or group which the function controls will not be output for the first record. If this is not the first record being output, all fields listed must have changed in order for the function to succeed.
- 3 The report formatter uses an OR operation between the listed fields, and succeeds if this is not the first record being output.

### Side effects

This function can be applied in conjunction with any other conditional function, such as `<if-unchanged()>`. The operator between the different functions is always AND.

### Examples

#### Example 1:

This example stops the output of a box unless the field `well_name` has changed. As we are outputting a `<ff>` function if the function succeeds, we do not want this to fire for the first record being output.

```
<box at b(*)+1,1  
<if-changed(well_name,2)>  
<t=<ff>>  
>
```

#### Example 2:

This example outputs the name of a company (in field `company`) and the corporate contact associated with it if either has changed.

```
<box at b(*)+1,1
```

```
<if-changed(company, contact, 1)>  
company <t= _/ > contact  
>
```



## <IF-EMPTY>

### Description

Controls the output of a box or box group, depending on whether a field or a set of fields is empty. If so, the box or box group will be output. If any of the fields listed have content, the box or box group will not be output.

### Scope

Layout box and box group function

### Syntax

```
<if-empty(field_name[, field_name...][, 1])>
```

where field\_name is the name of a field, a text variable reference or a comma-separated list of fields whose contents are to be checked for emptiness. Each field name can be qualified using subfield and/or part record nomenclature such as:

```
.2.field_name.5.2
```

which would test the emptiness of the second sentence of the fifth paragraph of the field in the second part record.

'1' is an optional flag which directs the formatter to apply an OR operation between the fields, rather than the default AND. If this flag is given, any of the fields being empty will result in the function succeeding.

### Side effects

This function can be combined with any of the conditional or <for> loop functions within a single box or box group. In this case, the operator between the various functions is always AND.

### Examples

#### Example 1:

This example suppresses the output of a box if the field speaker has content.

```
<box at b(*)+1,1  
<if-empty(speaker)>  
<t=Speaker is empty...>  
>
```

#### Example 2:

This example produces the following box if speaker, person or chapter are empty.

```
<box at b(*)+1,1  
<if-empty(speaker, person, chapter, 1)>  
...  
>
```

## <IF-NONEMPTY>

### Description

Controls the output of a box or a box group, depending on whether a field or a set of fields has content. If so, the box or box group will be output. This function is the logical complement to the <if-empty> function.

### Scope

Layout box or box group function

### Syntax

```
<if-nonempty(field_name[, field_name...][, 1])>
```

where field\_name is the name of a field, a text variable reference or a comma-separated list of fields whose contents are to be checked for non-emptiness. Each field name can be qualified using subfield and/or part record nomenclature such as:

```
.2.field_name.5.2
```

which would test whether the second sentence of the fifth paragraph of the field in the second part record had content.

'1' is an optional flag which directs the formatter to apply an OR operation between the fields, rather than the default AND. If this flag is given, any of the fields having content will result in the function succeeding.

### Side effects

This function can be combined with any of the conditional or <for> loop functions within a single box or box group. In this case, the operator between the various functions is always AND; i.e. all of the conditions being tested must succeed for the box or box group to be output.

### Examples

#### Example 1:

This example suppresses the output of a box if the field speaker does not have content.

```
<box at b(*)+1,1  
<if-nonempty(speaker)>  
<t=Speaker has the following content.../>  
speaker  
>
```

#### Example 2:

This example produces the following box if speaker, person or chapter have content.

```
<box at b(*)+1,1  
<if-nonempty(speaker, person, chapter, 1)>  
...  
>
```

## <IF-UNCHANGED>

### Description

Controls the output of a box or box group, depending on whether the contents of a field or a list of fields have changed since the previous record output. If said field contents have changed, the box or box group is suppressed. Optionally, the first record being output can count as unchanged or as changed (see flags).

*Note:*

*This function does not support subfield comparisons; thus all tests for difference are performed upon the first subfield or sentence of the field.*

### Scope

Layout box or box group function

### Syntax

```
<if-unchanged(field_name[,field_name...][,flags])>
```

where field\_name is the name of a field or a list of comma-separated fields whose contents are to be tested against their values during the last record output, and flags is an optional integer which can take the following values:

- 1 Instead of applying an AND operation between the fields given, the report formatter will use an OR; that is, if any of the fields in question are unchanged, the function succeeds. If this is the first record being output, the function will not succeed.
- 2 If this is the first record being output, the function will succeed; i.e. the box or group which the function controls will be output for the first record. If this is not the first record being output, all fields listed must be unchanged in order for the function to succeed.
- 3 This flag uses an OR operation between the listed fields, and succeeds if this is the first record being output.

### Side effects

This function can be used in conjunction with any of the other conditional or <for> loop functions within a single box or box group. In this case, the operator which is applied between the various functions is an AND operation, i.e. all of the functions must succeed in order for the box or box group not to be suppressed.

### Examples

#### Example 1:

```
<
  <box at b(*)+2,1
  <t=Person= >
  person
>
  <box at b(*)+1,1
  <t=Speaker= >
  speaker
```

```
>  
<box at b(*)+1,1  
<if-unchanged(speaker,person,2)>  
<t=   Record <rid> --- speaker and person   didn't  
change.>  
>  
>
```

This example shows all fields for speaker and person. If there was no change to either person or speaker since the previous record output, a comment to that effect is provided.



## <INDENT>

### Description

Directs the report formatter to indent the second through nth lines in the current box or box group by a certain number of columns. This can be useful when the box being output includes a label plus field content. If the field content stretches to two or more lines, these subsequent lines should be output so that their content lines up with where the content started on line 1—i.e., after the label.

### Scope

Layout box or box group function

### Syntax

```
<indent(n)>
```

where n is the number of columns by which the second through last lines are to be indented from the side of the box.

### Side effects

If a TEXT field which has been stored with 'Layout Retained' is output in a box which makes use of the indent function, the <noorig> function should also be used. If not, the data being output will be wrapped in the same way as it appears in the BAF, which could be very confusing, or at least unattractive.

### Examples

#### Example 1:

This report:

```
<
<box at b(*)+1,1
<indent(8)>
<t=Label : >
field_content
>
```

produces results such as:

```
Label : This is the field's content. As you can
see, it stretches over a single line. However, the
second and subsequent lines do not start flush
with the left hand side, but rather are indented
by eight columns.
```

#### Example 2:

This report is for the demonstration database Alice:

```
<
<box at b(*)+2,1
<t=Record <rid> content 'txt' in ALICE:>
>
<box at b(*)+1,1
<indent(6)>
```



```
<t=TEXT : >  
txt  
>  
>
```

and produces an output like this:

Record 1 content 'txt' in ALICE:

TEXT : Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

etc.



## <LINK>

### Description

The report filter <link> provides an easy way to load data from another database.

### Scope

Layout box or box group function

### Syntax

```
<link (link_fld.x, database, src_fld.y, sea_mode,  
dum_fld)
```

where link\_fld.x represents the field and subfield in the current database, which will be used as the record name for the lookup in the source database—hence identifying the required record. Database represents the source database, and src\_fld.y the field and subfield in the source record to be picked up and displayed here.

#### Link databases without record name fields

In order to access a link database without record name field and to load a PHRASE field, a non-exact search has to be made in the link database. This is indicated by a fourth argument (sea\_mode) with the following values:

- 0 : Search in the record name field in the link database  
(default)
- 1 : Search in all TEXT fields
- 2 : Search in all TEXT/PHRASE fields
- 3 : Search in all PHRASE fields
- 13 : Search for an exact match in all PHRASE fields
- field\_name : Search in the specified field field\_name
- 'field\_name' : Search for an exact match in the field field\_name

#### Retrieving a TEXT field

In order to load a TEXT field from a link database, a dummy TEXT field in the target database must be defined and passed to the link filter as a fifth argument (dum\_fld). The link filter must also be called at the very start of the report template (output format), i.e. before all box definitions. The dummy TEXT field will be loaded with the complete TEXT field from the link database and can then be output by the report generator in the usual manner.

#### Retrieving data from all records hit by a search

When retrieving several subfield hits the filter writes the data from all hits into a dummy field in the database record being formatted, much like the handling of getting the full TEXT field content from a link database. In order to use this, you have to define such a field for the database that is being output.

A problem is how to handle the transfer of data from the link database if the field to retrieve data from has a) many subfields or b) is a TEXT field. If this is the case, this dummy field must be a "part field". Otherwise, only the first (or single) subfield from the link database will be loaded.

E.g. <link(F1,linkdb,load\_fld,L1,dummy1)> will load data from all hits.

If the dummy1 field is a "part field", all the data from the load\_fld of the first hit will be stored in the dummy1 field of the first part record. If the dummy1 field is a regular field only the first subfield of load\_fld will be loaded into the first subfield of the dummy1 field.

The data from the second hit will be stored into part record two, and so on.

To summarize retrieving hits from several record hits:

- Needs a dummy field to store the data
- If the source field is TEXT or has several subfields, the target dummy field must be a part field of the same type.
- Target dummy field is not a part field
- Data from each record found is stored in each subfield
- Target dummy field is a part field
- Data from each record found is stored in each part
- Use standard formatting features to show the contents of the dummy field, either looping over subfields or both parts and subfields

### Side effects

None

### Examples

#### Example 1:

```
<box at b(*)+1,1  
<link(speaker.1,linkbase,anyfield.1)>  
>
```

This example performs a record name search in database Linkbase for the content of the first subfield of speaker, and then outputs the content of the first subfield of anyfield from the record found in Linkbase (if any).

#### Example 2:

Use the <link> filter to search for the content of field F1 in the L1 field of a link database:

```
<  
<box at b(*)+1,1  
<link(F1,link_db,load_fld,L1)>  
>  
>
```

#### Example 3:

Use the <link> filter to search for the content of field F1 for an exact match in the L1 field of a link database:

```
<  
<box at b(*)+1,1  
<link(F1,link_db,load_fld,'L1')>  
>  
>
```

**Example 4:**

Use the <link> filter to search for the content of field F1 for a match in all TEXT and PHRASE fields of a link database:

```
<
  <box at b(*)+1,1
    <link(F1,link_db,load_fld,2)>
  >
>
```

**Example 5:**

Using the <link> filter for a TEXT field:

Assume an existing dummy TEXT field of name dummy1 exists in the target database. *Note:*

*Exact matching using a field name has been chosen in the following example.*

```
<
  <link(F1,linkdb,load_fld,'L1',dummy1)>
  <box at b(*)+1,1
    <t=Hello world!>
  >
  <box at b(*)+1,1
    <t=Here comes the text from the link database.../>
  <s.p=/>
  dummy1
>
>
```

## <NOFF>

### Description

This function turns off the automatic formfeed printed when page size has been defined. Form feeds specified using `<T=<FF>>` will still work.

### Scope

Format function

### Syntax

`<noff>`

### Side effects

`<noff>` must appear at the beginning of the report specification, before the first box definition. It will not affect a page size definition stored in a printer definition file; it will affect only the page size within the report itself.

### Examples

#### Example 1:

```
<page size 60*132
<noff>
<box ...
>
>
```

## <NOLF>

### Description

This function turns off the automatic line planning.

This function also keeps whitespace in text inserts also for non-TEXT values. If you wish to compress whitespace sequences, you should also use <NOORIG> in your output format where relevant.

### Scope

Format function

### Syntax

```
<nolf>
```

### Side effects

<nolf> must appear at the beginning of the report specification, before the first box definition.

### Examples

#### Example 1:

```
<nolf>
<box at ...
>
>
```

## <NOORIG>

### Description

If a TExt field has been declared in the database design as being 'Layout Retained' (i.e. all spaces, blank lines etc. are maintained in the BAF), the <noorig> filter can be used to force stripping of such elements during output. This can be especially useful if the output box is to be smaller than the data entry box, or if the <indent()> function is used.

### Scope

Layout box or box group function

### Syntax

```
<noorig>
```

### Side effects

None

### Examples

#### Example 1:

This example outputs a layout-retained TExt field if using <indent()>. If <noorig> is not used:

```
<
  <box at b(*)+1,1
  <indent(8)>
  <t=Label : >
  field_content
>
>
```

and the data is, for example:

```
The fat cat from Jubaliyah's called up to Slim Jim
with an epithet of animalistic intent on his lips:
Hey lazy dog, call your sister for me.
```

then the output could look like :

```
Label : The fat cat from Jubaliyah's called up
to Slim
Jim with an epithet of animalistic
intent on
his lips:
Hey lazy dog, call your sister
for me.
```

As you can see, the original line breaks and spaces are maintained in the output - to the detriment of the appearance of the data. Whereas, using the <noorig> function:

```
<
  <box at b(*)+1,1 size* 46
```

```
<noorig>  
<indent(8)>  
<t=Label : >  
field_content  
>  
>
```

the output would be:

```
Label : The fat cat from Jubaliyah's called up  
to Slim Jim with an epithet of  
animalistic intent on his lips:  
Hey lazy dog, call your sister for me.
```





## <NUMFORM>

### Description

This function edits NUmber and INteger values. Headers, separators, and trailers apply as if the fieldname was entered alone.

<Numform> takes several arguments:

- field name, with or without a subfield number
- length in characters
- number of decimal positions
- a character specifying a formatting convention.

The first argument is the name of the field containing the value to be formatted.

The second argument can be zero, or any integer from one to the maximum page width. If zero is used, TRIP adopts the current box size as the length in characters.

The third argument can be any number from zero to the maximum page width.

The fourth argument is a value of one to three or the character 'e'. If this argument is an odd number (one or three), the output will be divided into groups of three, each group separated either by a comma [,] or full stop [.]. If the argument is an even number (two) or an alpha character ('e'), the output will not be grouped.

The default setting outputs ungrouped digits using the decimal point, for example, -12345.67.

Each number is output right-adjusted. If there is output overflow, where the number to be output is too long for its specified string, the string is filled with pound or number signs [#]. An empty subfield produces an empty string.

### Scope

Text string function

### Syntax

```
<numform(fieldname[.x],length[,precision[,format]]
)>
```

### Side effects

None

### Examples

#### Example 1:

```
<
<box at b(*)+1,1
<numform(n1,4)>
<numform(n1,4,0)>
<numform(n2.3,8,3)>
>
>
```

The first two are equivalent, where the integer values of the NUmber field n1 are output right-justified in strings that are four characters in length. The third example outputs the value of the third subfield of the NUmber field n2 in strings of eight characters, with three decimal positions.

Assuming the value of 'N2.3' to be -12345.67, several examples are listed with the output they produce in the table below.

<Numform> Statement	Output	Effect of Fourth Argument
<numform(n2.3,10,2,e)>	-1.23E+4	E denotes output in normal scientific notation
<numform(n2.3,10,2,1)>	-12,345.67	1 causes output of integers in groups of three digits, using the decimal point
<numform(n2.3,10,2,2)>	-12345,67	2 specifies no grouping of output, using the decimal comma rather than decimal point
<numform(n2.3,10,2,3)>	-12.345,67	3 causes output of integers in groups of three digits and specifies the use of the decimal comma rather than decimal point

**Table 6–13 Samples of <Numform> output**

## <OCCS>

### Description

Returns a text string containing the number of hit occurrences produced by the last search performed.

### Scope

Text string function

### Syntax

`<occs>`

### Side effects

None

### Examples

#### Example 1:

To output the number of hits in a search:

```
<
<box at b(*)+1,1
<t=The last search had <occs> hit terms.>
>
>
```

giving output of, for example:

```
The last search had 2578 hit terms.
```

## <ONCE>

### Description

If placed in a layout box, that box will only appear once per database in the resulting output, during the display of the first record.

See also <AT\_BEGIN>, which is similar but also works if the output will include data from more than one database.

### Scope

Layout box or box group function

### Syntax

```
<once>
```

### Side effects

None

### Examples

#### Example 1:

To output a cover page for printed output:

```
<
  <box at b(*)+1,1
  <once>
  <t=This print was produced on <curdate>.>
>
...
>
```

## <ORIG>

### Description

Gives an override in an individual box for a box group level <noorig>. Thus, if one TExt field out of many being output by a box group is to be output using its 'Layout Retained' feature, you should use <orig> in a specific box for that field.

### Scope

Layout box or box group function

### Syntax

```
<orig>
```

### Side effects

None

### Examples

#### Example 1:

To output three TExt fields which are all 'Layout Retained', but only one of which should be output in such fashion.

```
<
<
<noorig>
<box at b(*)+2,1
<indent(8)>
<t=Label : >
text_field_1
>
<box at b(*)+2,1
<orig>      ! Override <noorig>
text_field_2
>
<box at b(*)+2,1
<indent(8)>
<t=Label : >
text_field_3
>
>
>
```

which might output such as:

```
Label : This is data which was stored with the
Layout Retained attribute, but is being output
with that attribute suppressed.
```

```
This is data from a Layout Retained TExt field
which is being output with that attribute still in
place :...
```

```
As you can see, blank lines and
```

spaces are maintained.

Label : This is more data from a Layout Retained  
TExt field which has its layout attribute  
suppressed.



## <PAGENO>

### Description

Returns a text string containing the current page number, relative to the start of the current output.

### Scope

Text string function

### Syntax

<pageno>

### Side effects

None

### Examples

#### Example 1:

To output the page number at the top of every page:

```
<
  <header_box size 2*15
  <t=Page # <pageno>./>
>
>
```

which might give output such as:

```
Page # 1.
... actual data ...
<ff>
Page # 2.
... actual data ...
```

## <PARTS>

### Description

Returns a text string containing the number of part records which are associated with the meta-record currently being output.

### Scope

Text string function

### Syntax

<parts>

### Side effects

None

### Examples

#### Example 1:

To output a simple count of part records:

```
<box at b(*)+1,1  
<t=There are <parts> part records.>  
>
```

which might give output such as:

```
There are 5 part records.
```



## <RID>

### Description

Returns a text string containing the unique record number of the current record being output.

### Scope

Text string function

### Syntax

```
<rid>
```

### Side effects

None

### Examples

#### Example 1:

Output the number of the current record:

```
<box at b(*)+1,1  
<t=This is record number <rid>.>  
>
```

which might give output such as:

```
This is record number 42.
```

## <RIS>

### Description

Returns a text string containing the index of the current record within the last search performed. This index is an ordinal number between one and the maximum number of records hit by the last search.

### Scope

Text string function

### Syntax

```
<ris>
```

### Side effects

None

### Examples

#### Example 1:

Output the number of the current record and its index within the last search performed:

```
<box at b(*)+1,1  
<t=Record # <rid> is <ris> within search.>  
>
```

which might give output such as:

```
Record # 32658 is 5 within search.
```

## <RNAME>

### Description

Returns a text string containing the unique record name of the current record, if applicable.

### Scope

Text string function

### Syntax

<rname>

### Side effects

None

### Examples

#### Example 1:

Output the name of the current record:

```
<box at b(*)+1,1  
<t=This is record: <rname>.>  
>
```

which might give output such as:

```
This is record: CY93/1234
```

## <SORTFIELDS>

### Description

Allows a sort order to be embedded within the report itself. This sort order consists of a comma-separated list of field names from the database being output.

The <sortfields()> option must be placed before the first box is specified.

### Scope

Format function

### Syntax

```
<sortfields(field_name[, field_name...])>
```

### Side effects

This function effectively disables the SORT modifier in CCL. Any attempt to use Show or PPrint SORT will result in an error message. If data is being output from a database cluster, the data cannot be merged unless the user issues a DEFINE MERGE command.

### Examples

#### Example 1:

A statement to sort output on the contents of the Corr fields day, rname and raddr:

```
<sortfields(day, rname, raddr)>
```

## <SUBRID>

### Description

Returns a text string containing the unique number of the current part record being output.

### Scope

Text string function

### Syntax

<subrid>

### Side effects

None

### Examples

#### Example 1:

To output the current part record's unique number:

```
<box at b(*)+1,1  
<t=This is part record # <subrid>.>  
>
```

which might give output such as:

```
This is part record # 32.
```

## <SUBSTRING>

### Description

Returns a string extracted from a field, or subfield, where the extraction is defined by a start position and a length.

### Scope

Text string function

### Syntax

```
<substring(field_name, start_position, length  
[, justification])>
```

where field\_name is the field or subfield from which the string is to be extracted, e.g. rname.1.

Start\_position is the position within the field or subfield from which the extracted string should begin. This position is based on 1 being the first character in the field, or subfield.

Length is the number of characters to extract from the field or subfield in question. If start\_position + length is greater than the available data, the resultant string will be padded with blank (ASCII 32) characters. If length is zero (0), the extracted string will contain all characters from start\_position to the end of the field or subfield in question.

Justification is an optional argument, and can only take a single value, the case-insensitive r. This argument specifies that the extracted string should be right-justified within the output. For example, if length specifies twenty but the available data only stretches to ten characters, those ten characters would occupy positions eleven through twenty in the resultant string.

### Side effects

None

### Examples

#### Example 1:

```
<substring(p1,10,20)>
```

If p1 is a PHrase field, this function will create a string of twenty characters from the first subfield of p1, beginning with the tenth character of the subfield and padding the string with spaces to the right (if the subfield is less than twenty-nine characters long).

#### Example 2:

To create a field header using substring:

```
<h.field=<substring(p1.2,10,0)>>
```

Whatever information is contained in the second subfield of p1 from the tenth to the last character will be output in the field header.

#### Example 3:

To create a text variable using substring:

```
<2=<substring(t1.2.3,1,8)>>
```

Supposing t1 to be a TExt field, the first eight characters of the third sentence of its second paragraph will be loaded into the text variable <2>.

## <Text Variables>

### Description

A text variable can hold any number of characters, and can be used in any place where you can use a text string. The text variables are <0>, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8> and <9>.

### Scope

Format function

### Syntax

```
<variable=content>
```

```
<t=This is a use of a text variable : <variable>.>
```

where variable is an ordinal number between zero and nine inclusive, and content is any text string, which may contain any text string scoped function.

### Side effects

The text variables must be assigned before any boxes are coded within the format, otherwise their use will be ignored by the formatter.

### Examples

#### Example 1:

Loading, and using, a text variable:

```
<
  <1=TRIP Systems International, Inc.>
  <box at b(*)+1,1 <t=<1>... Yo!>
>
```

which would give output such as:

```
TRIP Systems International, Inc.... Yo!
```

#### Example 2:

Loading a text variable using a text string scoped function:

```
<
  <1=<substring(field1.2, 10, 10, r)>
  <header_box size 1*80
  <t=The contents of field1 are ... <1> >
>
>
```

which might have output such as:

```
The contents of field1 are ... cy93/1234
```

## <TRACE>

### Description

This function causes the search history leading to the last performed search to be output as it would appear in the search history window. This is useful for tasks such as producing print job cover sheets.

### Scope

Layout box or box group function

### Syntax

```
<trace>
```

### Side effects

None

### Examples

#### Example 1:

To print a sample cover sheet:

```
<
<box at b(*)+1,1
<trace>
<t=<ff>>
>
>
```

which could produce output such as:

```
S=1   <475> BAsE ALICE
S=2   <28>  Find mad hatter
```



## <TIMEFORM>

### Description

Returns a string containing a time value formatted as specified by the time format argument.

### Scope

Text string function

### Syntax

```
<timeform(time_value, time_format)>
```

where time\_value is the value which is to be formatted by the function. This value can either be field content, or the function <curdate>, which will yield the current time in twenty-four hour notation.

Time\_format is an integer value having the following meanings (default is 1):

**Time Format 1** If the time value does not include minute or hour segments, they will appear as zeros, e.g.:

Time Value	Sample Output
1hr, 10min	1:10:00
59 seconds	0:00:59

**Time Format 2** If the time value does not include an hour segment, the output will also not include an hour segment; i.e. only the minutes and seconds will be output by the function. If the time value does not include a minute segment, it will appear as a zero:

Time Value	Sample Output
1hr, 10min	1:10:00
59 seconds	0:59

**Time Format 3** If the time value does not include an hour or minute segment, the output will not include them:

Time Value	Sample Output
1hr, 10min	1:10:00
59 seconds	59

### Side effects

None

### Examples

#### Example 1:

Output the current time:

```
<
<box at b(*)+1,1
<t=The time is <timeform(<curdate>,1)>.>
>
>
```

which will produce output such as:

```
The time is 15:20:35
```

## <WEIGHT>

### Description

Returns a text string containing the value that the relevance rank engine has associated with the current record, with assigned rankings normalized to a percentile range. This function has no meaning unless the report in which it is used is invoked following a fuzzy logic search (FUZZ).

### Scope

Text string function

### Syntax

<weight>

### Side effects

None

### Examples

#### Example 1:

To output the rank of the current record:

```
<box at b(*)+1,1  
<t=Record <rid> has weight <weight>.>  
>
```

which would produce output such as:

```
Record 36 has weight 97.
```

## Chapter 7: Search Forms

The search form is an easy and straightforward alternative to CCL searching that does not require knowledge of command language syntax. It is a single page form where the user types the search expressions into search fields, which the search form then uses to generate a search order for TRIP to perform. The hit record counts of all the fields combined are shown in the form report line. There may also be individual hit record counts shown for each field. A database administrator may create several search forms for one or many databases, which can be linked together.

To view the existing search forms in TRIP, select the 'Search Forms' icon in the mmc window. A list of search forms will then appear.

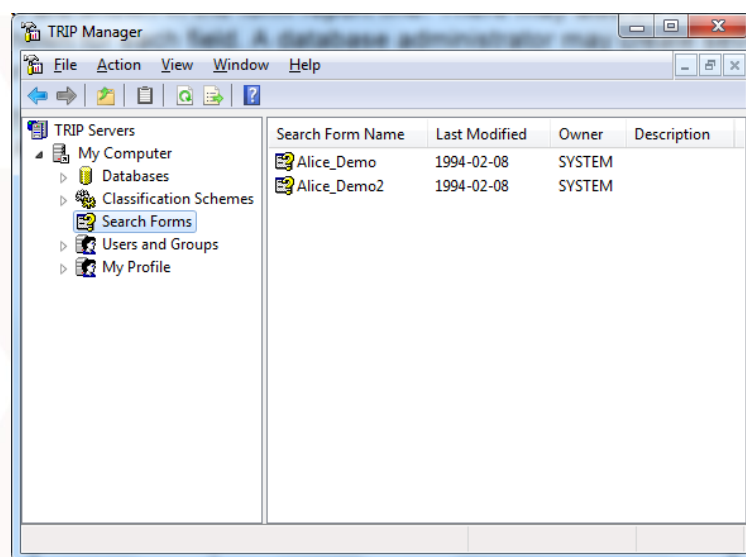


Figure 7–1 Search forms for a TRIP installation

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

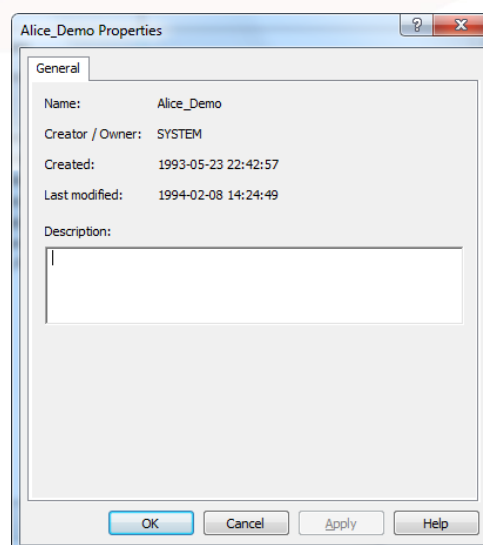


Figure 7–2 Properties for search form ALICE\_DEMO

## Creating and Modifying TRIPclassic Search Forms

Unfortunately, TRIPmanager currently has no means of carrying out these operations. It is hoped to include this functionality in a later release. For now, consult the TRIPclassic user guide for details of how to carry out these tasks.

### Copying TRIPclassic Search Forms

To copy a search form, click on 'Search Forms' in the chosen TRIP server sub-tree, select the form to copy and select 'Copy' from the action menu. Next, click anywhere in the right-hand pane of the mmc window to *deselect* the currently selected search form and select 'Paste' from the action menu.

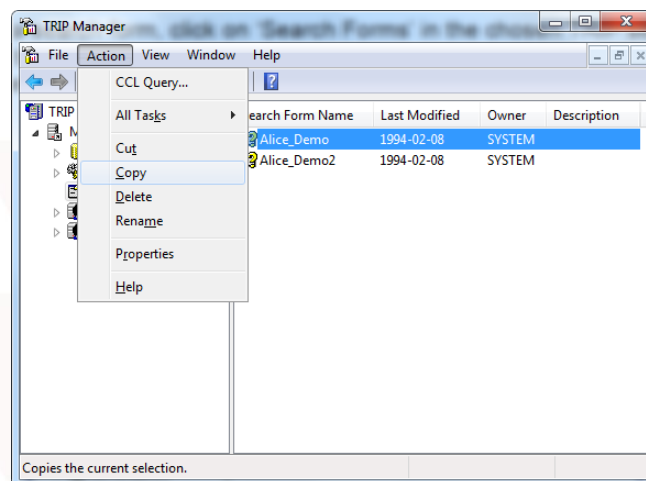


Figure 7-3 Copy a Search Form

A dialogue will appear as below, in which you may enter the name for the new copy of the Search Form and click on the 'OK' button to confirm the action.

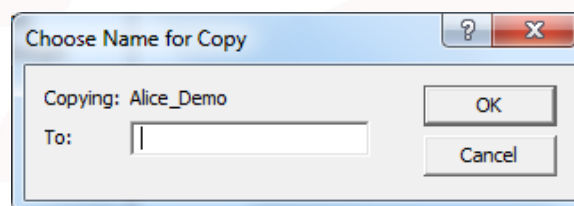


Figure 7-4 Name Search Form Copy

the new Search Form creation confirmation will then appear.

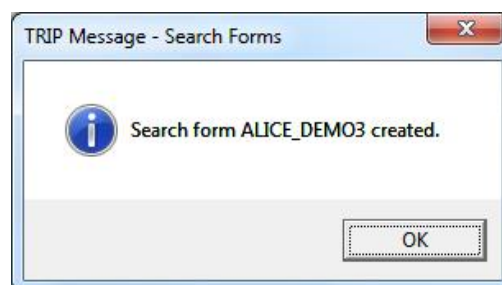


Figure 7-5 Search Form copy confirmation

Clicking on the 'OK' button will clear the confirmation. The copy of the form has now been created.

## Deleting TRIPclassic Search Forms

To delete a data entry form, click on 'Search Forms' in the chosen database sub-tree, select the form to copy and select 'Delete' from the action menu.

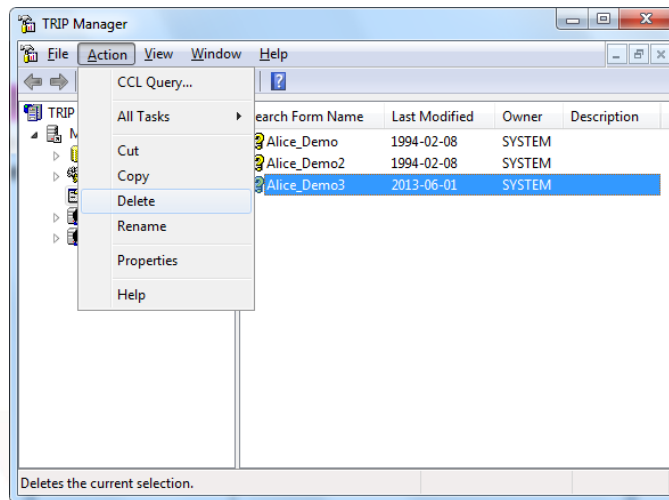


Figure 7-6 Delete a Search form

A 'Yes/No' confirmation dialogue will appear. Click the 'Yes' button to confirm the action, or the 'No' button to cancel it.

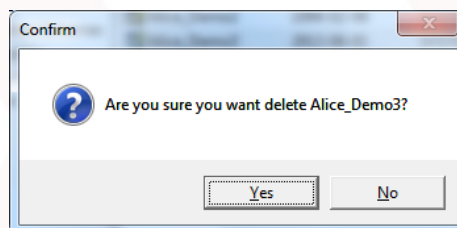


Figure 7-7 Delete Search Form confirmation

Clicking on 'Yes' will cause a deletion confirmation to appear.

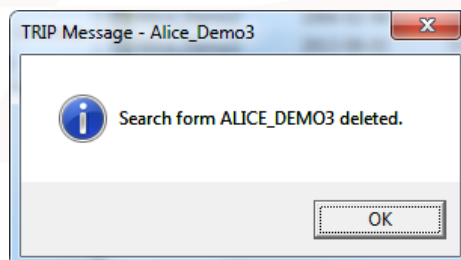


Figure 7-8 Search form Deleted

Clicking on the 'OK' button will clear the confirmation. The selected form has now been deleted.

## Part 3:

### Batch Update



## Chapter 8: Global Updating

*Note:*

*The command “UPDate SCoPe” cannot be found in this section because, despite having a similar name, it is not connected with global updating. For more information on this particular command, see “Appendix B – Scope Search Facility”, on page 273 of this manual and also the relevant section of the CCL Command Reference.*

Global updating is a means of making identical changes to a group of records, using a single updating order from the CCL command line. Using a set of records that have been identified either by a CCL search or a record number list, it is possible to:

- insert, delete or replace a field, subfield, sentence or paragraph
- delete or replace a string referred to by a search result in all the records of that search result, or insert a word or several words before it
- delete entire records, both those referred to by a search result and those identified by a list of record numbers.

As with manual data entry, update orders affect only the BAF of the database—the index files BIF and VIF remain unchanged until the database is indexed. Since searches are made using the index files and Show orders use the BAF records, if the BAF has been modified and remains unindexed, the user of the database will get conflicting results when searching in and showing the changed records.

Among other things, FOCus and Hlghlight may behave incorrectly when BAF records have been modified.

### Command Overview

An order for global updating has four main elements:

“Do this”	the type of update
to	
“This constituent”	the target of the update
with	
“This value”	the information to be changed
and	
“These records”	the records to be changed

as outlined below.

Command Element	Update Type	Update Target	Update Value	Update Domain
Explanation	What should be done?	What portion or constituent of each record will be changed?	What data or value is needed?	Which records will be altered?
Examples	INSert, UPDate or DELeTe	Field, Subfield, Word, Sentence etc.	XYZ, 123, etc.	WHere R=521, 687, 688, 1023, 2190, 2349 or WHere S=3

**Table 8–1 Anatomy of a global update command**

In TRIPclassic the maximum expanded order length is 400 characters. A longer order will lead to an error message. In applications created using the newer TRIPjxp and TRIPnpx APIs, there is no such limit.

*Notes:*

- *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.*

Only one database at a time can be open for updating, and referring to search results obtained from more than one database in the same update order is not permitted. You will need to index a globally-updated database before making local modifications or proceeding with another global update.

## Updating Using Record Numbers

### Command Structure

The four main elements of a global update by record number are summarized in the table below. Each is discussed separately in the sections that follow.

Update Type	Update Target	Update Value	Update Domain
INSert UPDate DELeTe	Record Fieldname.Subfield Fieldname.Paragraph Fieldname.Paragraph.Sentence Part.Fieldname.Subfield Part.Fieldname.Paragraph Part.Fieldname.Paragraph.Sentence	aaabbbcc 112233	R=record number

**Table 8–2 Structure of a global update using record numbers**



## Update Type

The three commands used are:

- INSert (short form: INS) to add a unit to the records
- DELeTe (short form: DEL) to remove a unit from the records
- UPDate (short form: UPD) to replace a unit, or, if the unit referred to is empty, add a unit.

## INSert Orders

INSert expands both structure and content of records. When a subfield, paragraph or sentence is inserted, the numbering sequence of any subfields, paragraphs or the sentences which follow is advanced by one position. For example, if a new 'Subfield 1' is inserted in a field, the existing 'Subfield 1' becomes 'Subfield 2', 'Subfield 2' becomes 'Subfield 3', and so on.

## UPDate Orders

UPDate takes the same arguments as INSert, replacing the unit referred to. Be very careful to provide complete specifications when using UPDate, to avoid errors such as replacing the contents of an entire field with a single phrase.

UPDate operates as an INSert order when there is nothing to replace.

## DELeTe Orders

DELeTe is somewhat different from INSert and UPDate orders, in that it is possible to delete the contents of entire records, part records and fields.

When a subfield, paragraph or sentence is deleted, the numbering sequence of any subfields, paragraphs or the sentences which follow recedes one position. For example, if you delete 'Subfield 1', 'Subfield 2' will become 'Subfield 1', and so on.

*Note:*

*To avoid the accidental deletion of the complete contents of a database, the global record delete has a default maximum value of 100 records that can be deleted in one go. This maximum value can be redefined: For more details, see the 'Define – Delete' section of the CCL Command Reference.*

## Update Target

When the records you want to make changes in are identified by a list of record numbers, your order must state by name and number what component of the record you want to change.

The shortest forms of the generic update targets are listed below.

Target	Shortest Form
Record	R
PART	PART

**Table 8–3 Generic update targets**

## Update Value

There are several basic restrictions regarding values that may be written to a field during global updating. The source and target field types must match,

i.e. it is not possible to insert text into a NUmber field. If the reserved word 'where' appears in the string to be introduced, the entire string must be enclosed in double quotation marks—we recommend the use of quoted strings as a matter of course for all TExt and PHrase target fields.

### Update Domain

The update domain consists of TRIP's reserved word WHere (short form: WH) followed by a series of record numbers (short form for Record: R) or a search result pointing to a subset of the records in the database (short form for Search Result: S). The domain in the first case consists of a sequence of record numbers, which follows the same rules as do other lists of numbers in TRIP. Numbers and number intervals must be separated by commas, must be listed in ascending order and cannot overlap.

There are three ways to state an interval:

- TO a number
- a number TO a number
- FROm a number

The first option, 'TO a number' must appear first in a mixed-interval CCL statement, and the third, 'FROm a number' must appear last to prevent record number overlap.

The order part stating which records to edit has the same form for INSert, DElete and UPDate and refers to the record numbers in the database itself. These take the format 'R=' followed by a list of record numbers or record number intervals, as seen in the examples below.

Example 1:

```
WHere R=to 10, 15, 20
```

which locates the first ten records in the database, as well as records fifteen and twenty.

Example 2:

```
WHere R=3, 5, 7 to 9
```

which locates record numbers three, five and seven through nine.

Example 3:

```
WHere R=FRom 1
```

which locates all the records of the database.

### INSert Examples Using Record Numbers

As before, most examples use the demonstration database Corr, and only one database is open at a time.

Example 1:

```
INSert rcomp="Paralog U.K." WHere R=70 to 74
```

appends a new subfield containing the phrase 'Paralog U.K.' to Corr's field rcomp in records seventy to seventy-four.

Example 2:

```
INS rcomp.1="Paralog U.K." WH R=75
```

inserts a new first subfield containing 'Paralog U.K.' to the field rcomp in record seventy-five. Existing data in the field will be moved one subfield forward.

Example 3:

```
INS content.3.1="I told you so." WH R=FR 95
```

inserts a new first sentence to the third paragraph of the TExt field content of records ninety-five upwards in the database. Existing sentences in the paragraph will be moved one step forward.

### UPDate Examples Using Record Numbers

Example 1:

```
UPDate scomp="Paralog U.K." WHere R=70 to 74
```

replaces the contents of the field scomp, (even if it contains several subfields) in records seventy through seventy-four with a single subfield containing the phrase 'Paralog U.K.'.

Example 2:

```
UPD scomp.1="Paralog U.K." WH R=75
```

replaces the contents of the first subfield of scomp with 'Paralog U.K.' in record seventy-five.

Example 3:

```
UPD content 3.1="I told you so." WH R=fr 95
```

replaces the first sentence of the third paragraph of TExt field content with 'I told you so.' in records numbered ninety-five and above.

### DELeTe Examples Using Record Numbers

Example 1:

```
DELeTe R WHere R=to 3, 6, FRom 98
```

deletes records one through three, six and ninety-eight and above.

Example 2:

```
DEL saddr WH R=7 to 10
```

deletes the field saddr from records seven through ten.

Example 3:

```
DEL scomp.1 WH R=75
```

deletes the first subfield of scomp in record seventy-five.

Example 4:

```
DEL content.3.1 WH R=FR 95
```

deletes the first sentence of the third paragraph of content from record ninety-five onwards.

## Updating Using a Search Result

### Command Structure

The four main elements of a global update by search result are summarized as before.

Update Type	Update Target	Update Value	Update Domain
INSert UPDate DELeTe	Record Part (for use with DELeTe only) Field Subfield Paragraph Sentence Word Fieldname.Subfield Fieldname.Paragraph Fieldname.Paragraph.Sentence Part.Fieldname.Subfield Part.Fieldname.Paragraph Part.Fieldname.Paragraph.Sentenc e	aaabbbccc 111222333	S=search number

**Table 8–4 Structure of a global update using a search result**

### Update Type

As with updates using record numbers, the commands are INSert, DELeTe and UPDate.

### Update Target

When the records to be changed are referred to by a search result, the search result itself refers to a specific portion of the record. It is therefore not necessary to identify them by name and number, but only by their level of organisation (field, subfield, paragraph, sentence or word) within the record.

Target	Shortest Form
FIeId	FIE
SUBFIeld	SUBF
PARAgraph	PAR
SENtence	SEN
WORD	WORD

**Table 8–5 Record update targets**

### Update Value

The same restrictions apply here as for global updates with record numbers.

### Update Domain

The order part stating which records to edit has the same form for INSert, DELeTe and UPDate, and takes the form 'S=', followed by the number of a single search result:

WHere S=2

### INSert Examples Using Search Results

Example 1:

INSert content.3.1="I told you so." WHere S=4

inserts a new first sentence to the third paragraph of the TExt field content of records located in search number four. Existing sentences in the paragraph will be moved one step forward.

When inserting in the records of a search, you can refer to the position of the hits of the search, instead of referring to a field by name, as shown by the following examples.

If you are in any doubt about what positions a search refers to, you should look at its record first, using highlight. The highlighted units are the ones that your editing orders will use as reference points.

Example 2:

```
INS SENTence="I told you so." WH S=5
```

The phrase 'I told you so.' is inserted before each sentence that the search number five has found.

Example 3:

```
INS WORD="John" WH S=2
```

The word 'John' in either TExt or PHrase fields is inserted before each word that the search result two has detected.

Example 4:

```
INSert SUBField="John Smith" WH S=3
```

The phrase 'John Smith' is inserted as a new subfield before each subfield that search number three has found.

### UPDate Examples Using Search Results

Example 1:

```
UPDate content.3.1="I told you so." WHere S=4
```

replaces the first sentence of the third paragraph of the TExt field content with 'I told you so.' in the records of search number four.

Example 2:

```
UPDate SENTence="I told you so." WH S=5
```

replaces each sentence that search number five has located with 'I told you so.'

Example 3:

```
UPDate WORD="John" WH S=2
```

Replaces each word that search result two has found (in TExt or PHrase fields) with 'John'.

Example 4:

```
UPDate WORD="John Henry" WH S=2
```

replaces each word that search two has hit (in TExt or PHrase fields) with 'John Henry'.

Example 5:

```
UPDate SUBField="John Smith" WH S=3
```

replaces the contents of each subfield of a PHrase field pinpointed by search result three with the phrase 'John Smith'.

Example 6:

```
UPDate SUBF=1993-03-01 WH S=7
```

replaces the contents of each subfield located by search seven with the date 1993-03-01 (the hits are intended for a DATE field, but the contents would be accepted for a PHrase field as well).

Example 7:

```
UPDate PARagraph="Yours truly" WH S=5
```

replaces each paragraph that search number five has found with the words 'Yours truly'.

Example 8:

```
UPDate price.1=30 WH S=8
```

In a database containing a NUMBER field named 'Price', the value in its first subfield will be changed to 30 for all records located by search eight.

### DELeTe Examples Using Search Results

Example 1:

```
DELeTe content.3.1 WHere S=4
```

deletes the first sentence of the third paragraph of the TEXT field content from the records in search number four.

Example 2:

```
DELeTe content.4 WH S=4
```

deletes the fourth paragraph of the TEXT field content from the records in search number four.

Example 3:

```
DELeTe R WH S=5
```

deletes the records found in search number five.

Example 4:

```
DELeTe FIEld WH S=6
```

deletes the fields hit by the search number six from the records of that search.

Example 5:

```
DELeTe SUBField WH S=7
```

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

Example 6:

```
DELeTe (name, addr, phone).SUBF WH S=2
```

deletes a tuple, in this case the contents of those subfields of fields name, addr, and phone that have the same number as the tuple subfield found by search number two.

Example 7:

```
DELeTe PARagraph WH S=2
```

deletes the paragraphs located by search two (in TExt fields) from the records of that search.

Example 8:

```
DELeTe SENtence WH S=8
```

deletes the sentences pointed to by search result eight (TExt fields) from the records of that search.

Example 9:

```
DELeTe WORD WH S=3
```

deletes the words hit by search three (in TExt or PHrase fields) from the records of that search.

## Global Updating of Part Records

Global updating may also be used to delete, insert or update part records or portions of part records.

If a search result contains hits in part fields, use

```
DELeTe PART WHere S=2
```

to delete the record parts found in search result number two. If the search has detected only records containing head fields, nothing will be deleted.

*Note:*

*You must provide the word 'Part' in its entirety; otherwise TRIP may interpret it as an action on a PARagraph.*

You may use DELeTe, UPDate and INSert orders to edit the contents of part fields or their subfields or sentences pinpointed by search results as well.

To edit the contents of a field in a single record part, refer to the record part by its number in the record. For example,

```
UPDate 2.name.1="John Smith" WH R=10 to 20
```

will substitute 'John Smith' for the contents of the first subfield of name in the second record part of records numbered ten to twenty.

## Copying With Global Update

Records modified in global updating may be inserted in a database other than their database of origin, or be appended to the end of the source database. This is done by inserting the modifier Copy directly after the command word of an INSert, DELeTe or UPDate order.

Copying is done in the same manner as for EDit Copy orders, i.e. fields with the same names must be of the same field type in both target and source databases. Fields that exist in the source database but not in the target database will be discarded during copying.

### Examples:

#### Example 1:

```
BAS corr
DEfine COPY=corr1
```

The first order opens the copy source database Corr, while the second opens Corr1 as the copy destination.

#### Example 2:

```
INSert COPY WHere R=10,15 to 18, FRom 95
```

This order, following the two previous orders, copies the Corr records ten, fifteen through eighteen and ninety-five and above to Corr1.

#### Example 3:

```
Find taiwan (creates search result S=2)
UPDate WORD="china" WH S=2
UPDate COPY WORD="china" WH S=2
```

Both UPDate orders modify records that contain the word 'Taiwan', but the first of them stores the modified records in the source database Corr, and the second inserts them in the copy destination database Corr1.

#### Example 4:

```
INS COPY content="Copy extracted from CORR."
WH S=2
```

This order inserts a new paragraph containing 'Copy extracted from Corr.' in the field content of every record of search number two and writes the updated records into database Corr1.

#### Example 5:

```
DElete COPY=corr2 rname WH s=2
```

This order deletes the field rname from the records of search result two and adds them to copy destination Corr2. Corr itself remains unchanged.

#### Example 6:

```
DEfine COPY=corr
DElete COPY rname WH R=30 to 33
DElete sname WH R=30 to 33
```

Here the DEfine order makes the current database Corr the copy destination as well. The first DElete order deletes the field rname and appends records thirty through thirty-three to the end of the database as new records. The second order deletes the field sname from records thirty through thirty-three.

## Case Sensitivity

In some situations, global updating will convert lower-case letters to upper-case. If the character string in the updating order contains only lower-case letters and spaces, the following will occur:

- If the word hit consists of upper-case letters only, the string will be converted to upper-case before the exchange.



- If the first letter of the word hit is an upper-case letter, and the following letter is in lower-case, the first letter of the replacement string will be converted to upper-case.

## The Log File

An updating order puts a batch process in the queue, after some preliminary error checking. The resulting log file name is and  
GUdatabasename\_uniqueID.log.

It should be noted that while an updating job is placed in the queue immediately, a Print order (without a NOW, NO HOLD, or WAIT modifier) submits the job to the queue when the user leaves TRIP. This is important if you request a printout of records that you are going to delete in the same session.

## Error Checking

A great part of the checking of an order's correctness is done by the batch process, not by TRIP itself. This is meant to save the user time, because a single order may involve a lot of checking.

TRIP checks that the user has write access to the database, that the order is syntactically correct, and that it involves no immediate type clash (that is, that the changes intended for the first of the records referred to in the order are possible). An impossible change could be, for example, inserting a text string in a NUmber field.

Further error checking is done by the batch process, and the results are recorded in the log file mentioned above. If, for example, you make a mixed search order that found hits in both TExt and NUmber fields, and you attempt to delete a word from the fields found during that search, an error message will be written to the log file and no changes will be posted to the BAF.

The only mixed search result accepted in editing orders is one consisting of both TExt and PHrase references used in an editing order where words are inserted, updated or deleted.

This restriction is intended for error prevention; all checking is done before writing to the BAF.

## Chapter 9: Loading, Indexing and Reindexing

TRIP provides three utilities for automated data entry (loading) and database indexing (making the data searchable), 'Index', 'Load/Index' and 'Load'. All three utilities submit jobs to be executed in background mode in both UNIX and Windows.

### Index

Indexing renders data searchable by updating the index files BIF and VIF so that they conform to any changes (additions, alterations or deletions) that have been made to the BAF since the latest updating. This may be done by anyone who has write access to the database. To ensure that the index files are updated regularly, you may wish to organize the indexing if several persons are to be updating the database.

Choose 'Index' if you are using manual data entry to write information to the BAF.

To perform an index, highlight the database to be indexed, then select 'All Tasks' then 'Index' from the Action menu.

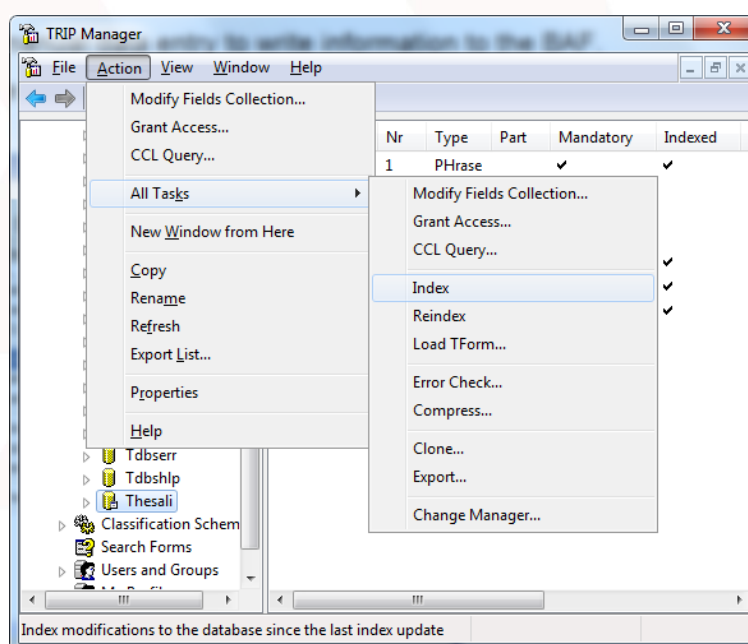


Figure 9–1 Indexing the Database TestThes

The CCL command INDEX may be used instead of the method above: E.g., the order:

```
INDEX thesali
```

given in the CCL command window, or as part of a TRIP procedure, begins a process which indexes the database Thesali.

## Load and Load/Index

Loading uses records from a file in TRIP's entry format TForm to update the BAF.

With 'Load/Index', automated data loading to the BAF is immediately followed by the updating (indexing) of the BIF and VIF.

To perform an TForm load, highlight the database into which the TForm is to be loaded, then select 'All Tasks' then 'Load TForm...' from the Action menu:

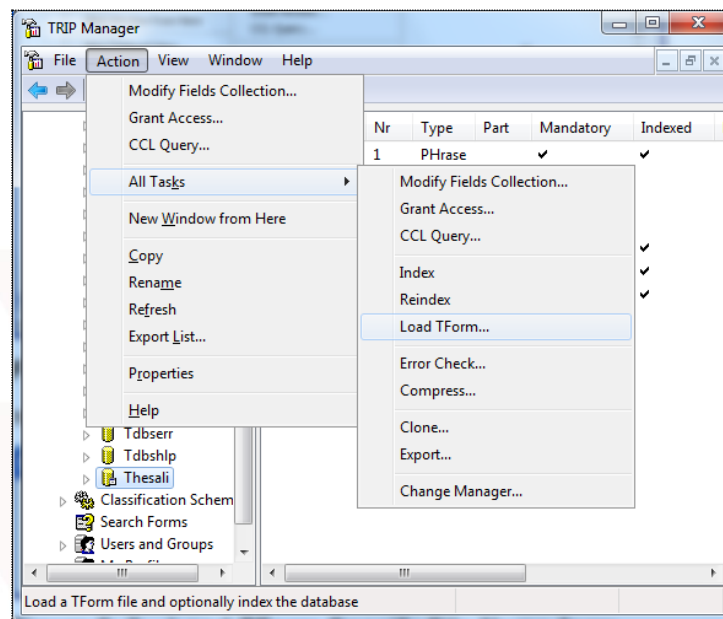


Figure 9–2 Load a TForm file into database TestThes

which will result in the appearance of the 'Specify File Name' form:

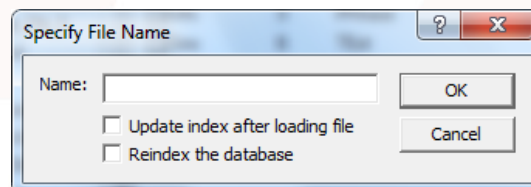


Figure 9–3 Load TForm Specify File Name form

To perform a simple 'Load', just enter the file path and file name into the 'Name' entry box and click on the 'OK' button.

If you wish to perform a Load/Index, check the 'Update index after loading file' checkbox.

Checking the 'Reindex the database' checkbox as well, will cause a reindex to be performed, rather than just a normal index.

*Note:*

*For more details on reindexing, see the section 'Reindexing a Database' later in this chapter.*

## Checking the Results

When a batch job is submitted, a log file is created containing the results of each step in the procedure. The log files are named as follows:

Operating System	Log File Name
UNIX/Windows	IXdatabasenameunique ID.log
	LIdatabasenameunique ID.log
	LDdatabasenameunique ID.log

Table 9–1 Operating systems and log file names

The log files will be placed in the directory from which the job was initiated. However, these can be rerouted to a different directory by defining the logical name TDBS\_LOG.

Remember to delete the log files, or set up a batch procedure to remove redundant log files.

## Error Logging

Error logging is the process of segregating all records which do not match the database design (contain illegal dates, patterns etc.) in a designated log file called ERRLOG\_databasename.TFO. If a record in a TForm file is not written to the BAF during loading, it contains an error and is written to the error log file in TForm instead.

The ERRLOG\_databasename.TFO file is normally placed in the area pointed to by the logical name TDBS\_LOG. If TDBS\_LOG is undefined, then ERRLOG\_databasename.TFO will be stored in the current directory, and if the area pointed to is not accessible, nothing will be added to the database.

*Note:*

*Some small deviation from the original format may appear in records written to the error log file. This is caused by intermediate storing in an internal format, and does not generally interfere with normal functioning.*

## Reindexing a Database

Under certain extreme conditions, it may become necessary to force the complete reindexing of a database in its entirety, rather than indexing only the most recent changes made. These include:

- modifying the indexing options for one or more fields in the database design; for example, from 'Index' to 'No Index'
- an index job has failed, the database has become corrupted and indexing is no longer possible; for example, not enough disk space has been allotted for temporary index file storage

If it becomes necessary to reindex all the records in the database, the BAF 'Indexed' markings may be removed by:

- Running the index command line program with the option "--reindex" that performs a bafini style initialization of the database prior to performing the actual indexing operation.

*Note:*

Type “*index --help*” for to get more information.

Operating System	Command
UNIX	\$TDBS_EXE/index --reindex
Windows	Index --reindex

- running a utility called BAFINI in the following manner:

*Note:*

Type “*bafini --help*” to get more information.

Operating System	Command
UNIX	\$TDBS_EXE/bafini
Windows	bafini

Table 9–2 Running the BAFINI utility

This places another marker in the BAF to notify TRIP that the database should be completely reindexed.

## When Batch Jobs Fail

### On UNIX and Windows systems

Success and failure log files are written to the location pointed to by TDBS\_Fel! Hittar inte referensälla. (See page **Fel! Bokmärket är inte definierat.** of this guide for more details). If TDBS\_LOG is undefined, any log files will be saved to the application’s current working directory, not the directory from which the application was started.

### On UNIX systems only

TRIP will generate e-mail messages when a batch job fails to complete. The message is sent to the initiator of the batch job, unless the logical name TDBS\_ERRMAILST has been defined (refer to the document “TRIPsystem Environment” for more information). The content of the message will describe the location of the log of the failed job.

## Part 4:

### Database Security



## Chapter 10: User Privileges

### TRIP's internal Access Privileges

When not using external logon verification, TRIP employs four levels of user privilege; the system manager, the database administrator (file manager) and user manager, the user group and the individual user.

#### The TRIP System Manager

Each TRIP installation has only one system manager, user identity SYSTEM. This person has complete system access privileges and rights, file and user manager as well as individual user.

The system manager creates the first user identities. You must be either the system manager or a user manager to give users or groups of users access to the TRIP system, and only SYSTEM can assign database or user managerial rights; that is, create a database administrator or user manager. SYSTEM is also the owner of all database administrators and user managers, regardless of their original creators.

#### The TRIP 'Superman' Logical Name

This logical name gives the TRIP system manager, SYSTEM, complete access to all TRIP objects. For more details, refer to the document "TRIPsystem Environment".

### TRIP File and User Managers

#### File Manager

The database administrator (or file manager, abbreviated FM) creates databases and assigns users and user groups access to them.

When a user is given manager rights, the ownership of this user is automatically transferred to SYSTEM.

There can be an unlimited number of database administrators per TRIP installation.

#### User Manager

The user manager (abbreviated UM) creates new users and user groups.

A user identity can be deleted only by the user manager (or system manager) who created that identity.

There is no limit to the number of user managers an installation may have.

#### The TRIP User Group

The user group concept enables the database administrator to give access rights to databases collectively. User groups are intended to simplify the administration of database access rights, and represent collections of users which have been granted a common and identical set of access rights to one or more databases.

For example, if one hundred users are registered in a TRIP installation, and ninety-three users must be able to read and write to five database fields while the remaining seven need the same access to all fifty fields of the same database, rather than create one hundred sets of individual access rights, an administrator can create two user groups.

A database administrator may grant access rights to his or her databases for individual users as well as groups. The creator of a group (the UM) may grant membership in the group to any individual user, regardless of whether this UM created the user in question or not. The combined access rights of each user are defined by the union of his or her individually-granted rights and the rights of the groups to which the user belongs.

Creating a group adds a new record to CONTROL, as does creating a new user. This record contains the group's access rights and the user names of its members.

Only a group's owner/creator may add a member to or delete a member from that group; however, the new member may have been created by another user manager.

Group membership is recorded in the user record of the individual user as well as that of the group. A group, like a user, can only be deleted by the UM who created it.

### **The Individual or End User in TRIP**

The number of possible users depends on the system site license purchased.



## Creating a New TRIP User

To create a new user, open the 'Users and Groups' sub-tree, then select 'My users'. Next, select 'New User...' from the action menu.

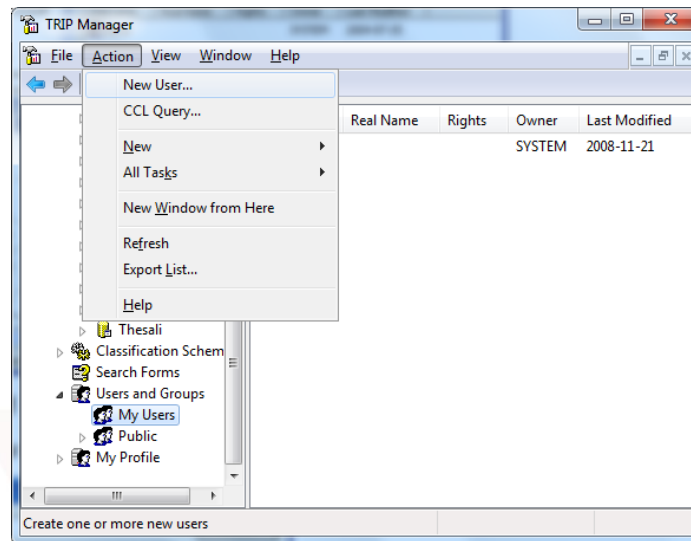
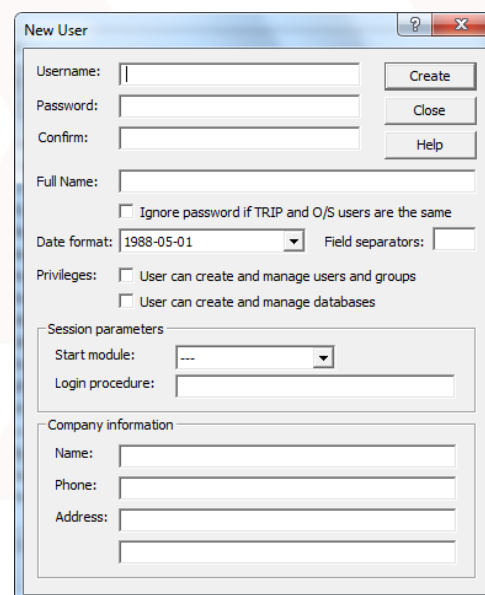


Figure 10–1 Creating a New User

this will cause the New User details form to appear:



The 'New User' form contains the following fields and options:

- Username:** Text input field.
- Password:** Text input field.
- Confirm:** Text input field.
- Full Name:** Text input field.
- ☐ Ignore password if TRIP and O/S users are the same
- Date format:** Dropdown menu showing '1988-05-01'.
- Field separators:** Text input field.
- Privileges:**
  - ☐ User can create and manage users and groups
  - ☐ User can create and manage databases
- Session parameters:**
  - Start module:** Dropdown menu.
  - Login procedure:** Text input field.
- Company information:**
  - Name:** Text input field.
  - Phone:** Text input field.
  - Address:** Text input field.

Buttons: 'Create', 'Close', 'Help'.

Figure 10–2 The create New User form

The minimum information required, is the username and password.

*Note:*

*Both user name and password may have a maximum of thirty-two characters.*

The password is not echoed (displayed to the screen), and the verification must match whatever has been entered as the password.

The other details that can be entered at user creation time are:

- Full Name

- Ignore Password if TRIP and O/S users are the same (See 'Appendix B: Local System Validation' for more details)
- Date Format
- Privileges
- Session Parameters
- Login Procedure
- Company Information

These items are covered in greater detail in the User Properties section of this chapter and in Appendix B: Local System Validation.

Once all desired details have been entered, you then create the user by clicking on the 'Create' button, causing the user created confirmation dialog to appear:

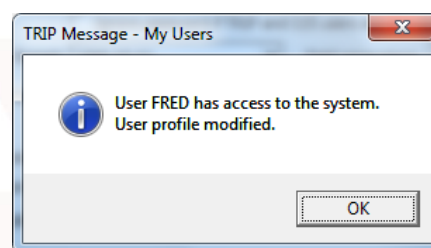


Figure 10-3 The User created confirmation dialog

This results in a new user record in CONTROL, where the access rights to databases, group membership, and manager privileges of the new user are stored. Whenever a user attempts some action within TRIP, his or her rights are checked against this user record, which determines what he or she will be allowed to do within the system.

## Deleting a TRIP User

To create a new user, open the 'Users and Groups' sub-tree, then select 'My users'. Next, select the user to delete and choose 'Delete' from the action menu.

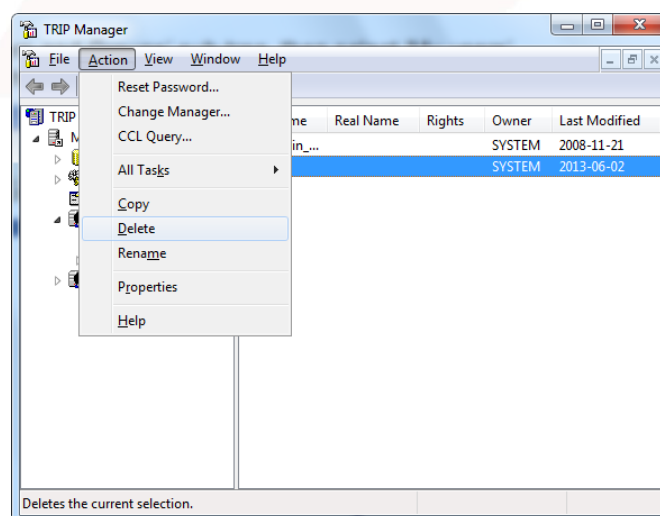


Figure 10-4 Deleting the user 'Fred'

A Yes/No confirmation dialogue will then appear:

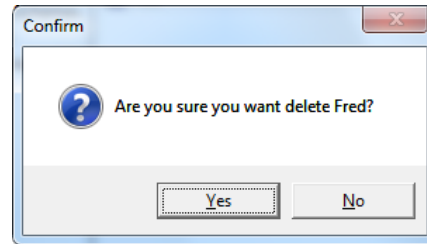


Figure 10-5 The Delete User Confirmation

Clicking on 'Yes' will delete the user, resulting in the confirmation shown below, whilst clicking on 'No' will abort the operation.

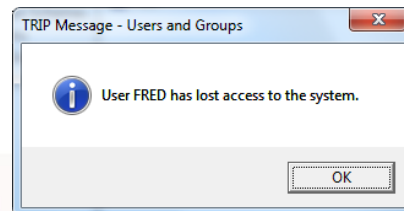


Figure 10-6 The Deleted User Access Loss Confirmation

Notes:

- The user SYSTEM cannot be deleted.
- Only the creator of a user can delete that particular user
- When a user is deleted, all database access rights and privileges for that user are also removed. Therefore, if the user to be deleted is a Database Administrator or a User Manager, you must transfer all of that particular user's holdings before they can be deleted.

## User Properties

To obtain the 'User Properties' window, open the 'Users and Groups' sub-tree, then select 'My users'. A list of users will be displayed.

Next, select the desired user and choose 'Properties' from the action menu.

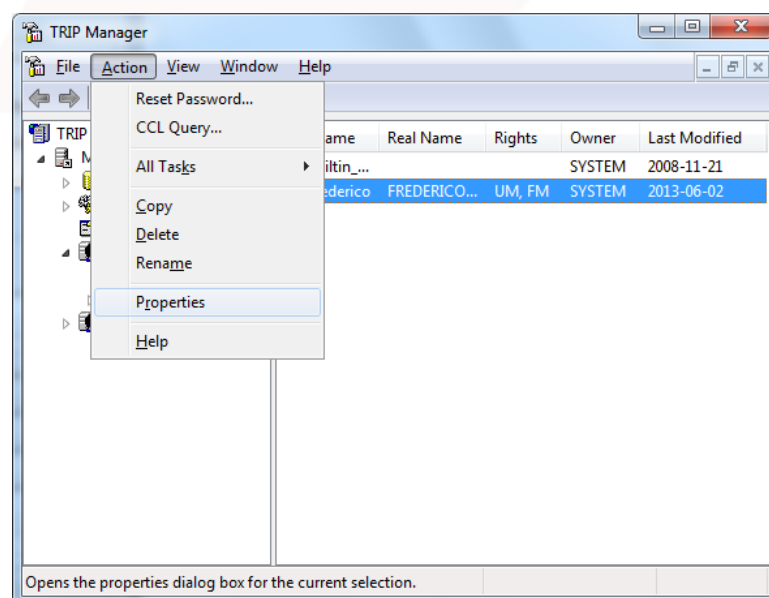


Figure 10-7 Opening Properties for the User, FREDERICO

The specific user's properties form will be displayed:

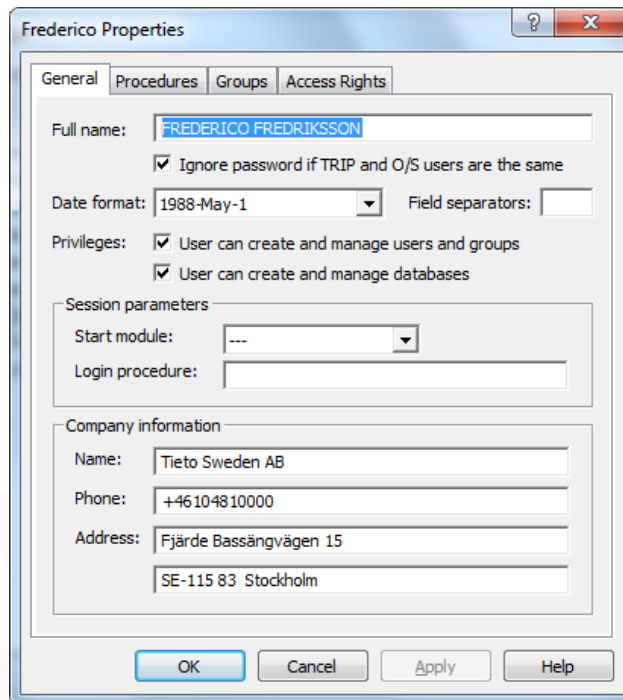


Figure 10–8 The user FREDERICO's user Properties form

The user manager who has created a user identity may enter data such as name and address, as well as some other TRIP defaults in the user's properties. These properties are covered in more detail overleaf.

## User Properties (1) – General

### Full Name

This is a descriptive entry to enable the user to be clearly identified. It will only be displayed in the administrator's console and may be a maximum of 255 characters in length.



Figure 10–9 Date Format selection box

### Ignore Password

A user may be permitted to enter TRIP without supplying a password, provided that their operating system and TRIP user names match. This, allows for the use of system logon validation in place of TRIP's own validation.



Figure 10–10 Ignore TRIP password checkbox

To grant this privilege, check the checkbox shown above in Figure 10–8.

### Date Form

Both the format and the separating characters of a user's current date form may be changed here. The date form in use is shown to the right of the phrase 'Date form:'.

To change the date form, use the drop-down selection box on the 'General' tab of the user's properties form:

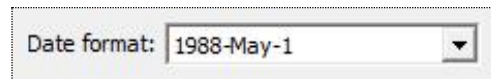


Figure 10-11 Date Format selection box

and select your desired date format:

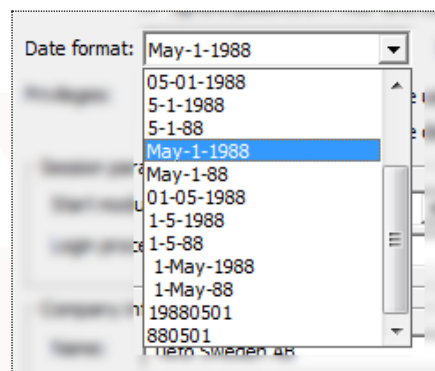


Figure 10-12 Date Format Selections

To change the digit separator characters for a date, use the 'Field separators' entry box, to the right of the date format selection list:



Figure 10-13 Changing the date digit separator

and enter the desired separator character, which should be one of the following: a hyphen [ - ], slash [ / ], full stop [ . ] or full colon [ : ].

The current date form is recognised in search orders (in addition to the system default), and output by the system in SStatus lists and in the output of records. A report may, however, specify some other date format.

### Management Privileges

This is where you can decide to grant or deny File and/or User Manager privileges to the user.

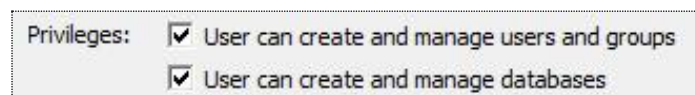


Figure 10-14 Management privilege settings

Selecting the top checkbox, 'User can create and manager users and groups', in the 'Privileges' section on the 'General' tab of the 'User Privileges' form, will grant 'User Manager' privileges to the user. Selecting the bottom checkbox will grant 'File Manager' privileges.

### Session Parameters

These are the parameters automatically used by the user's TRIP session when it starts at user logon time.


A screenshot of a 'Session parameters' form. It contains two fields: 'Start module:' with a dropdown menu showing '---' and a small downward arrow, and 'Login procedure:' with a text input field.

Figure 10–15 Session parameter settings

There are two parameters:

#### Start Module

This is only useful for users of the TRIPclassic product. Choose the appropriate value from the drop-down or leave it at its default value.

Selecting the Start module as 'CCL Search' deposits the user directly in the search order window after login, while 'Search Form' opens the default search form defined for a database.

#### Login Procedure

This value defines the name of a procedure (optionally qualified by the name of a group) that will be automatically executed whenever a TRIP application is run. Typically such procedures are used to establish defaults and aliases for common commands and modifiers.

#### Company Information

These fields are strictly informational and have no bearing on the well-running of a TRIP system. Use them for whatever seems appropriate.

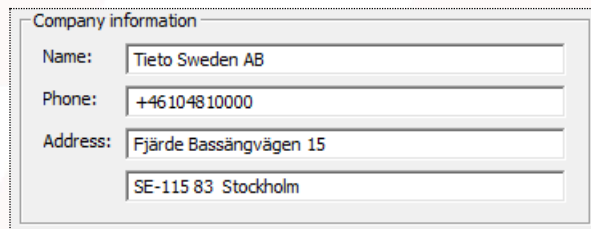
A screenshot of a 'Company information' form. It contains four text input fields: 'Name:' with 'Tieto Sweden AB', 'Phone:' with '+46 10 48 10 000', 'Address:' with 'Fjärde Bassängvägen 15', and a second line of the address with 'SE-115 83 Stockholm'.

Figure 10–16 Company information entry area

These (and other fields) are available for use by client applications via the TRIPclient and TRIPjtk 'User' object and also in the TRIPnpx and TRIPjxp 'TdbUser' class.

### User Properties (2) – Procedures

This property page is accessed by clicking on the 'Procedures' tab of the User properties form and is purely information as TRIP does not allow one user to modify the procedures of another user.

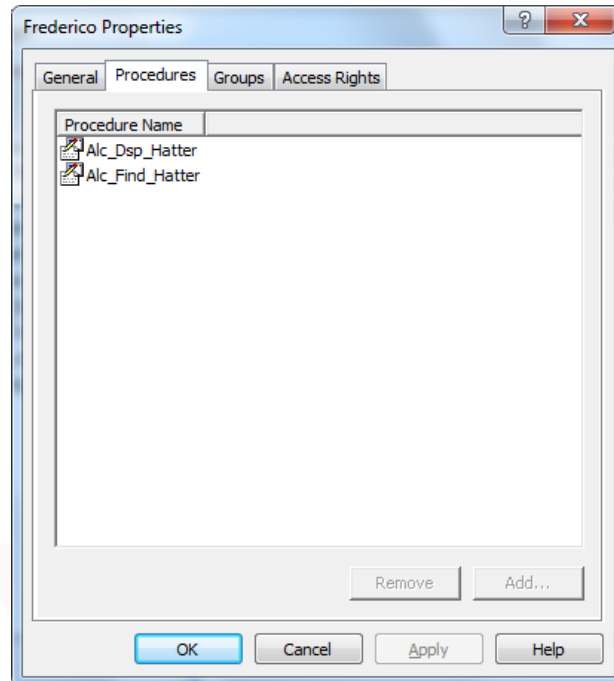


Figure 10–17 Procedures for user FREDERICO

### User Properties (3) – Groups

Clicking on the 'Groups' tab of the User Properties form will display the groups to which the particular user belongs.

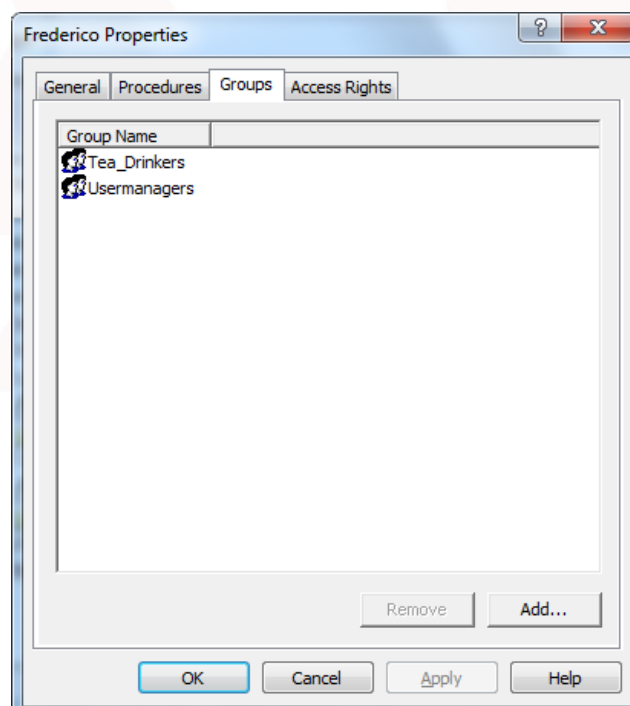


Figure 10–18 Group membership for user FREDERICO

The user, 'Frederico', being inspected above is already a member of 'Tea\_Drinkers' and 'Usermanagers'.

To remove a user from a group of which they are already a member, simply select the group name in the list and click on the 'Remove' button, which will no longer be greyed out.

To add the user to a new group simply click on the 'Add...' button, then select the desired group from the drop-down list on the 'Add To Group' form that appears:

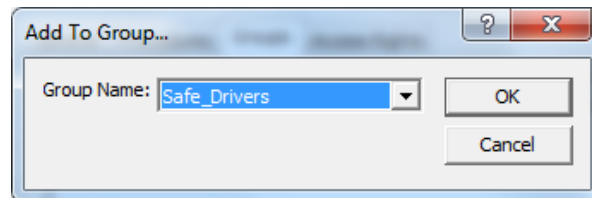


Figure 10–19 The Add To Group form

Clicking on the 'OK' button confirms your selection.

### User Properties (3) – Access Rights

Use this dialog to inspect and modify the individual (as opposed to group-inherited) access rights that one of your users maintains.

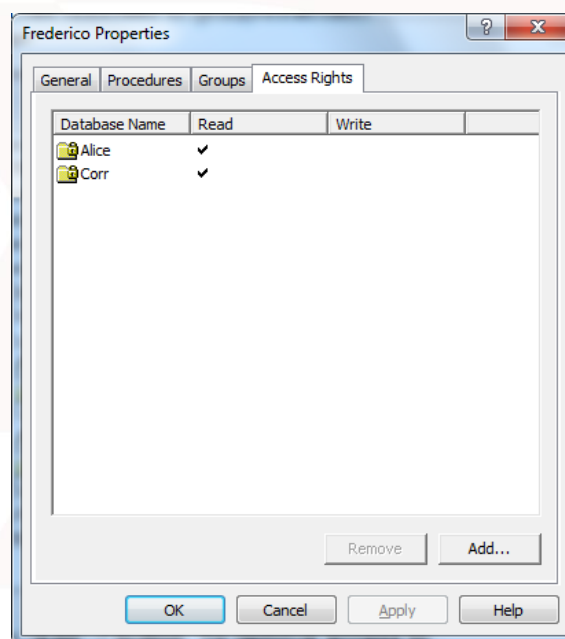


Figure 10–20 Access Rights for user FREDERICO

In order to add access to a new database, click the "Add..." button. To remove access to an existing database, select the database in the list and click the "Remove" button.

For more detail on access rights, see Chapter 11 of this guide.

### Creating a User Group

To create a new Group, select the 'Users and Groups' icon, then select 'New', 'New Group...' from the action menu.



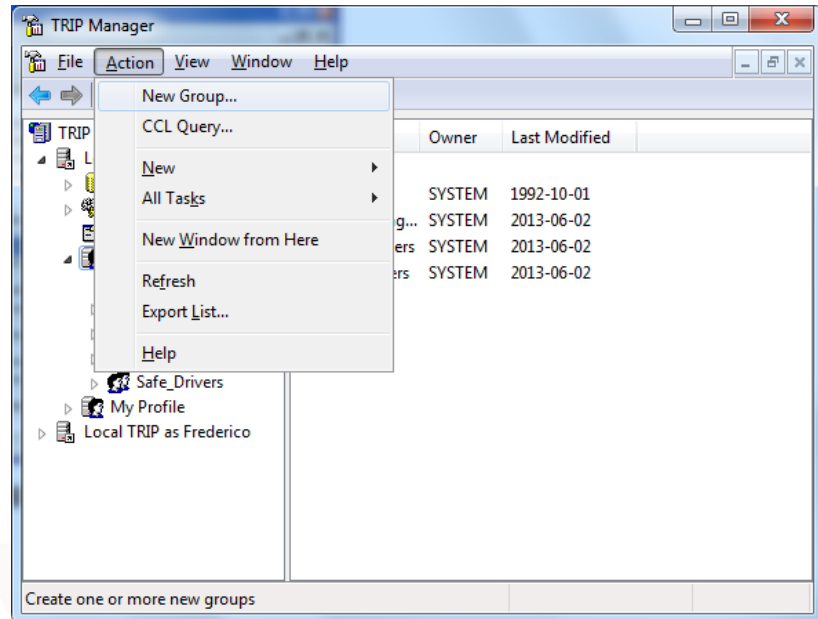


Figure 10–21 Creating a New Group

The 'New Group' name dialogue will be displayed:

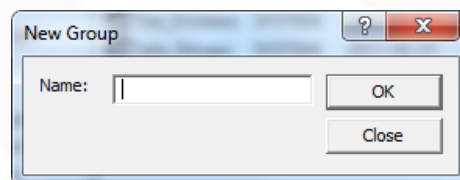


Figure 10–22 New Group dialogue

Enter the name of the new group to be created, then click on the 'OK' button. Creation of the new group will be confirmed with success message:

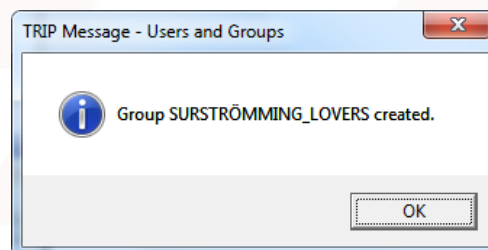


Figure 10–23 New Group Created Confirmation

## Deleting a User Group

To delete a Group, select the 'Users and Groups' icon, then select the group to be deleted. Next, select 'Delete' from the action menu.

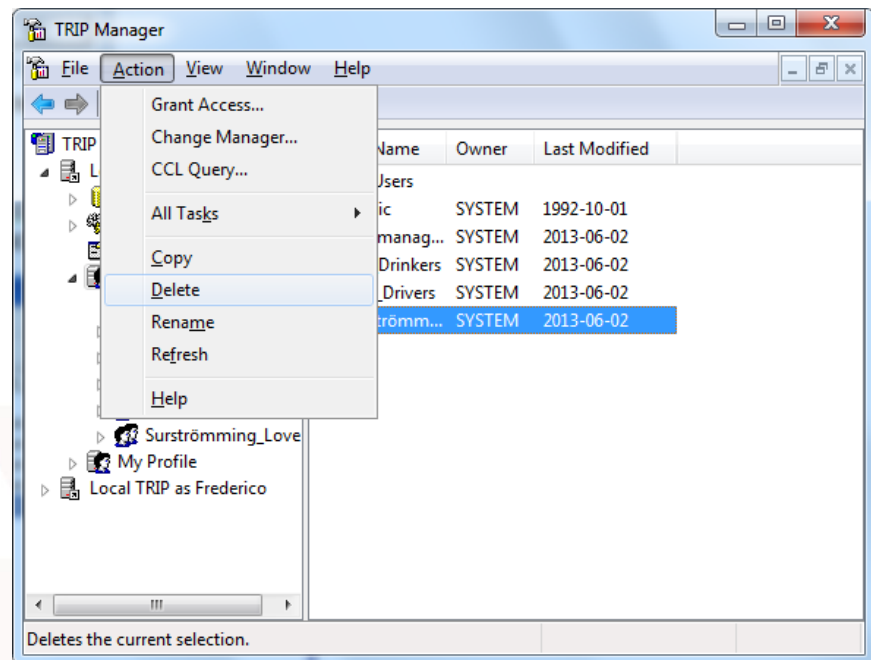


Figure 10–24 Deleting a group

A 'Yes/No' confirmation box will then appear:

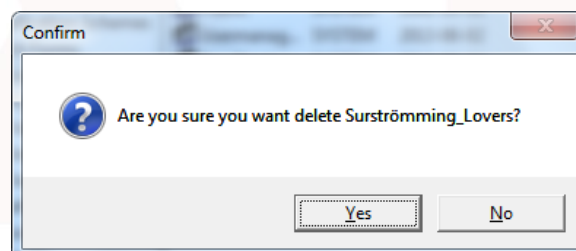


Figure 10–25 Confirming deletion of a group

Clicking on 'No', will leave the group untouched, while clicking on 'Yes' will confirm deletion of the selected group, after which a confirmation dialog will appear. The confirmation dialog can be cleared by clicking on its 'OK' button.

## Adding a Group Member

First, click on the 'My Users' sub-tree of the 'Users and Groups' icon to display the users managed by you

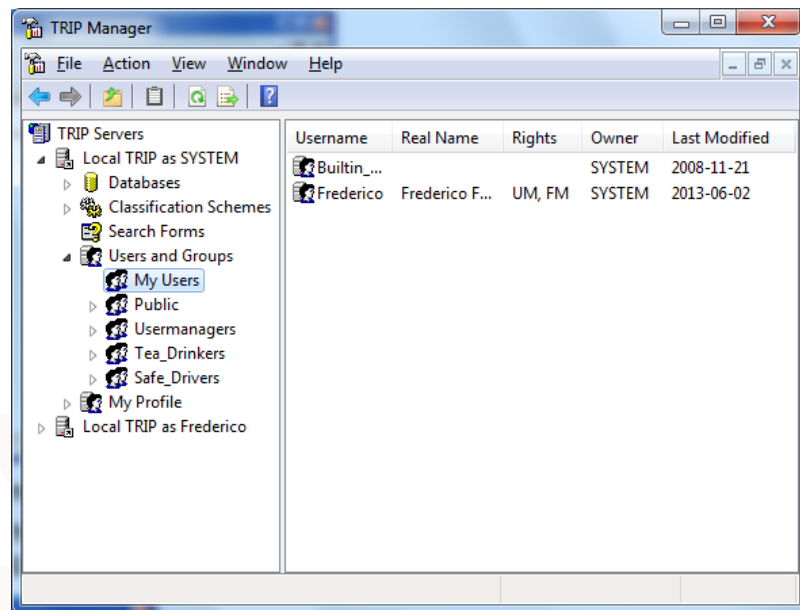


Figure 10–26 The 'My users' sub-tree

Next, drag and drop the selected user, from the right hand window to the desired group in the left hand window. A confirmation dialog box will appear to verify the action:

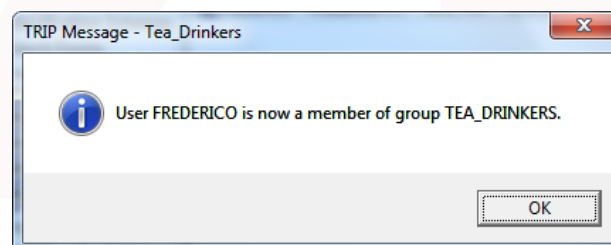


Figure 10–27 The Add Group Member confirmation

When a user is added to a group, he or she inherits any access rights assigned to the group as a whole.

## Deleting a Group Member

Select the desired group from which to delete the user; then select the user and chose 'Delete' from the action menu.

A 'Yes/No' confirmation box will then appear:

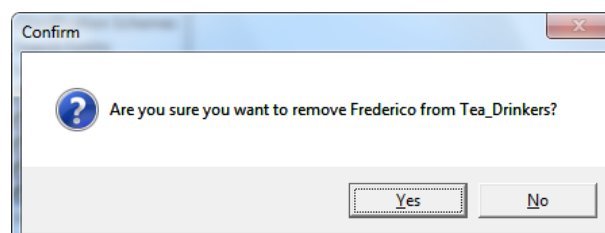


Figure 10–28 The Delete Member confirmation

Selecting 'Yes' will confirm the removal of the user from the group and a confirmation dialog will appear. The dialog can be cleared by clicking on its 'OK' button.

*Note:*

*The user will not be removed from the system. For details on how to remove a user, see the section entitled, 'Deleting a User'.*

## Transferring User Responsibility

A user manager may transfer the management of his or her users and/or user groups to another user manager in the system. This is done by selecting the user or group in question and choosing 'Change Manager...' from the action menu.

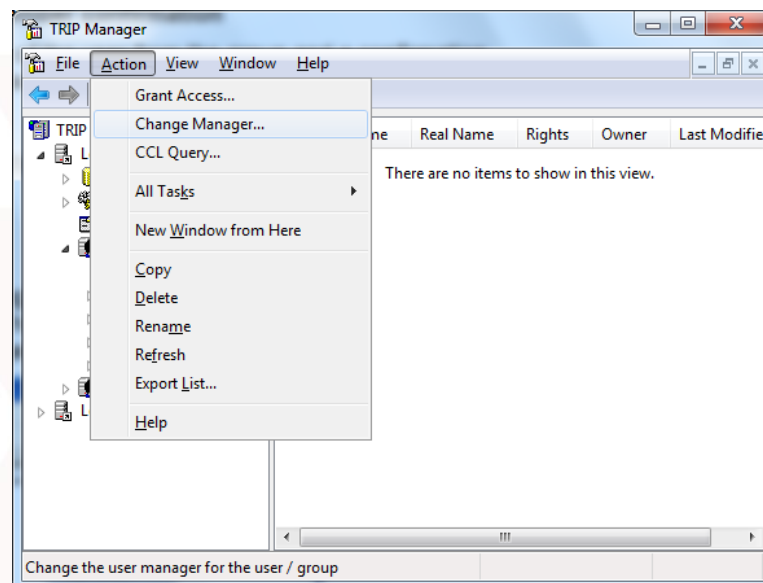


Figure 10–29 The Change Manager option

the 'Change Manager' selection box will appear:

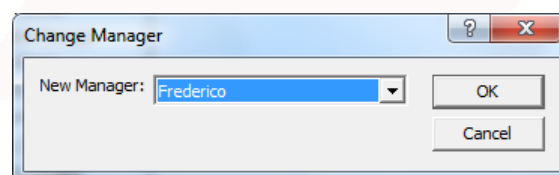


Figure 10–30 Change Manager Selection box

Select the desired new manager from the drop-down selection and click on the 'OK' button to confirm your choice.

The choice will be confirmed in the usual way, with a confirmation dialogue:

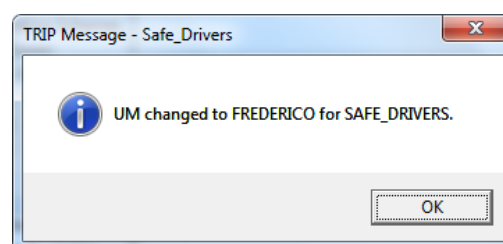


Figure 10–31 Change Manager Confirmation

The system manager may transfer any user or group to another user manager, regardless of their owners/creators, except for the group PUBLIC and the 'Change Manager...' option will not appear on the action menu if the action is not possible.

## Related CCL Commands

### Show

A user manager may use the order:

```
Show USer
```

to view a list of all of the users created by him or her with their database access rights, group membership and possible manager privileges

To obtain the corresponding information about groups, use the command:

```
Show GRoup
```

To list information regarding a particular user or user group, add the name of that user or group to the Show statement as follows:

```
Show USer R=George ↵  
Show GRoup R=Sales ↵
```

These Show statements request information about the user 'George' and the group 'Sales'.

A non-managerial user may obtain information in this way only for themselves, not for another user identity. The system manager can request an overview of all the users and groups in existence with the addition of R=ALL.

### Print

Use the corresponding Print orders to send the output to a file or printer:

```
Print user r=username ↵  
Print group r=groupname ↵
```

## Chapter 11: Access Rights

Read and write capabilities or access rights to a particular database may be granted only by the owner of that database; that is, its Database Administrator. Read access encompasses viewing rights only, while write access implies read access and includes append, alter and delete capabilities.

Access rights may be assigned not only to entire databases but to selected fields and/or selected records as well.

To assign access rights to a database, select the required database in the Databases tree, then select '\*Grant Access...' from the action menu:

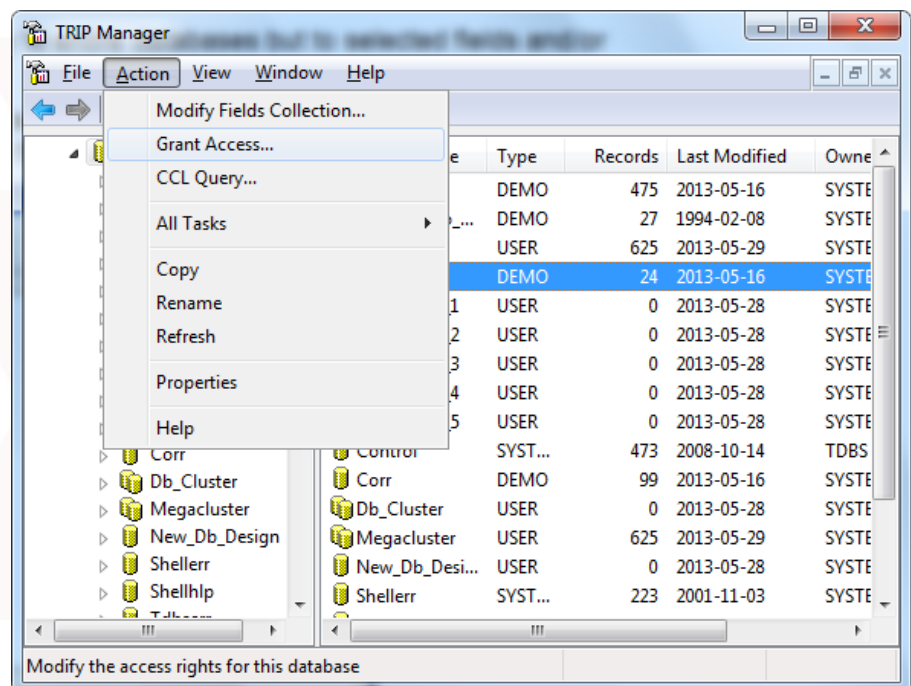


Figure 11–1 Granting Access to Database CARROLL

The Access Level form will appear, allowing you to set the access for the database,

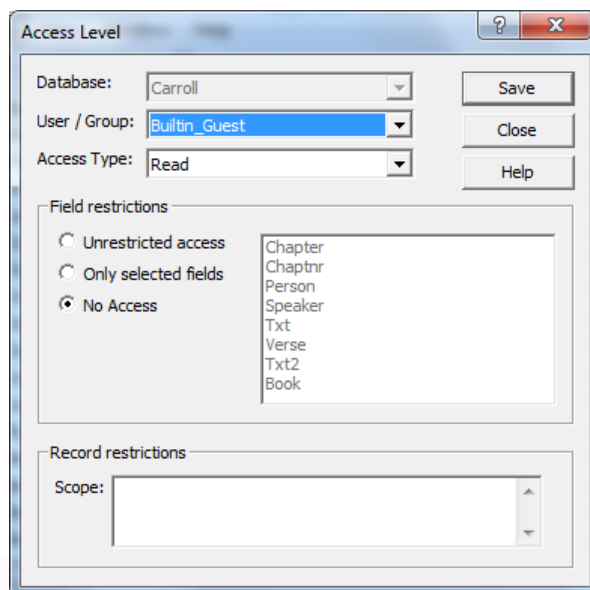


Figure 11–2 The Access Level Form

**Notes:**

- The system checks that you are the owner of the database; if you are not, then the 'Grant Access...' option will not appear on the action menu.
- You need not be a user manager or the owner of the user group in question to grant them access to a database. You must, however, be the author or creator of that database.

## Database Access Rights Definition

### Database

At the top of the Access Rights form, is the Database name selection box:

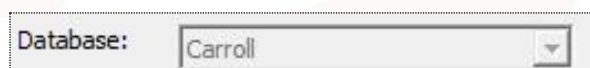


Figure 11–3 Database Name Selection

The database name will be automatically filled in on the access form and hence is not selectable.

### User / Group

Use this selection box to set which user or group you that wish to define access for.



Figure 11–4 Database Name Selection

### General Field Access

Field access is defined using a combination of the 'Field restrictions' and 'Record restrictions' sections on the lower half of the Access Restrictions form:

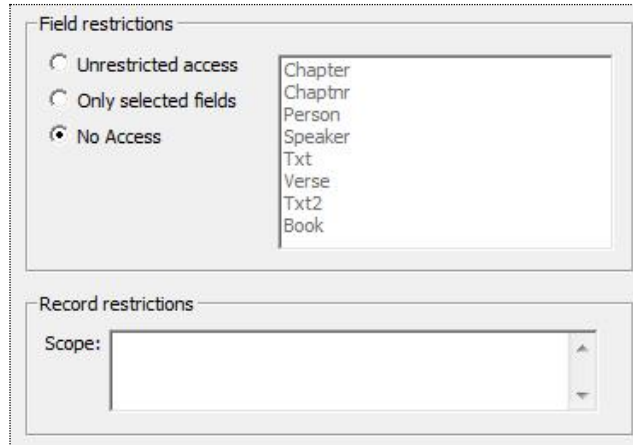


Figure 11–5 Field and Record Restrictions

Clicking on one of the three available radio buttons in the 'Field restrictions' section will allow you to set, or clear the current user, or group's, access settings.

If it is desired to further restrict access at the record level, enter the required access scope into the 'Scope' box in the 'Record restrictions' section of the form.

A user may be assigned almost any combination of the following access rights. These combinations are detailed in the tables overleaf.

The following Field Access combinations are possible:

Type of Access	Effect in Database
Access Type: READ Field restrictions: Unrestricted access	Allow read access to all fields
Access Type: READ Field restrictions: No Access	Disallow all read access
Access Type: READ Field restrictions: Only selected fields	Allow read access to chosen fields
If Read Scope is set	Allows read access to chosen records
Access Type: WRITE Field restrictions: Unrestricted access	Allow read and write access to all fields
Access Type: WRITE Field restrictions: No Access	Disallow write access
Access Type: WRITE Field restrictions: Only selected fields	Allow read and write access to chosen fields
If Write Scope is set	Allows write access to chosen records

Table 11–1 General field access rights



The following combinations are not possible:

Read Access	Write Access
No Access	Unrestricted access
No Access	Only selected fields
Only selected fields	Unrestricted access

Table 11–2 Unsupported combinations of access rights

The access form always shows the current state of the user's rights to a database.

*Note:*

*If an option is unavailable, it will be rendered unselectable.*

### Only Selected Fields Access

If 'Only selected fields' radio button is set, the fields list in the right hand pane will be available for the selection of specific fields. Those fields that are SELECTED will have their access set to the same access type that has been selected in the 'Access Type' box.

Holding down the <Ctrl> key on the keyboard whilst selecting, will allow multiple fields to be selected with the right mouse button.

Any or all fields and their contents can thus be hidden from view or protected from alteration by any or all users.

### Record-Level Access

You may restrict read and write privileges to selected records of the database by entering the arguments of a search order in the entry field 'Read Scope' (for read access) or 'Write Scope' (for write access) at the bottom of the 'Database Access Rights Definition Form'. The read scope (for searching and showing) and/or write scope (for data entry and modification) on the record level can be restricted using record numbers or field content.

For example, with READ access type selected, entering the CCL command fragment `'walrus OR carpenter'` into the 'Read restrictions' 'Scope' box, allows the user to read only those records containing the terms 'walrus' and/or 'carpenter' (a positive read scope):

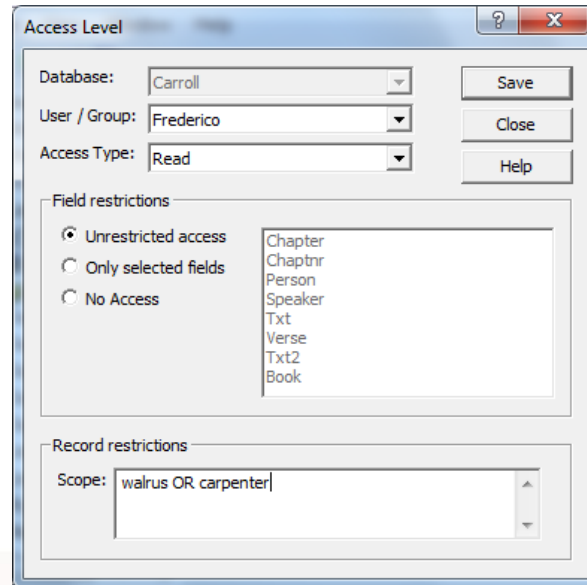


Figure 11–6 Record-level READ rights for 'FREDERICO'

and, with WRITE access type selected, entering the CCL command fragment `walrus or (carpenter AND alice)` into the 'Record restrictions' 'Scope' box allows modification of only those records containing 'walrus' or 'carpenter' and 'Alice':

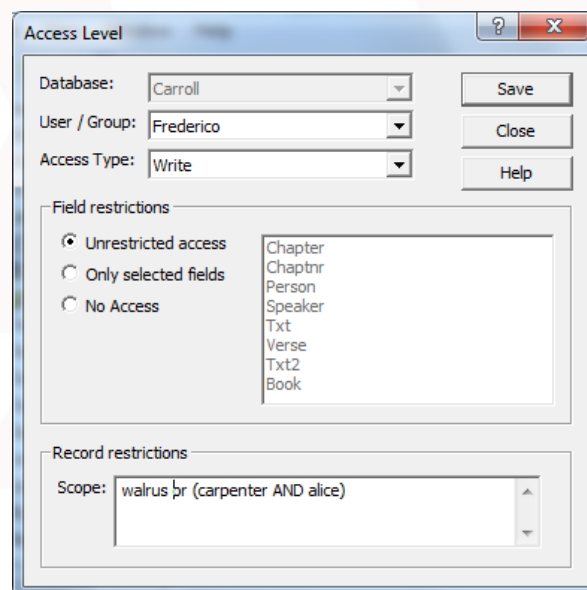


Figure 11–7 Record-level WRITE rights for 'FREDERICO'

Using the examples above, to restrict access to only those records which do not contain either 'walrus' or 'carpenter' (a negative read/write scope), prefix the command fragment with ALL NOT; i.e.

```
ALL NOT walrus OR carpenter
```

And

```
ALL NOT walrus or (carpenter AND alice)
```

You may restrict write access to selected records independently of record-level read access.

When you have completed the forms for this user, click on the 'Save' button to save the access settings. The user's access rights to the database will be stored in that user's properties.

When the user opens a database that he or she has restricted access to and the result of the BASE order is written in the search history window, the number of records is given as the records of the hidden read scope. Every search order is limited to those records in the same manner as in the DEFINE SCOPE order.

## The Hierarchy of Access Rights

If a user is granted access to a database not only as an individual, but as a member of one or more user groups, then his or her user rights will be the union (rather than the intersection) of what is granted.

For example, one particular user has been assigned SELECTED access to the demonstration database Corr, with the ability to read five fields and write to none. This user has also been made a member of User Group 1, which as a group has read/write access to twelve fields, User Group 2 with access to three fields, and User Group Public, with full read/write access. This user will in actuality possess complete read and write privileges to Corr regardless of the access rights assigned on the individual level, as the most liberal and inclusive combination of access rights possible always prevails.

A second user has been given SELECTED read access to fields one and two and group read access to fields three and four. His or her total (cumulative) access will be to fields one through four.

## Database Cluster Access

Access to a database cluster is granted in the same manner as with individual databases, with one important exception; at this level, a database administrator can only allow a user to know of the existence of the database cluster by granting 'READ - All' access. All other read and write access privileges must be assigned at the level of the individual database.

## About Read-Protected Fields

A 'hidden' or read-protected field is one to which a user has no read access.

### Hidden Fields and Searching

Scope checking during searching is bi-level, encompassing both pre- and post-search lookups.

To activate pre-search checking, the user must call a hidden field by name in his or her search order. If a read-protected field name is not so specified, the

search is performed, read privileges are checked and the now-filtered list of fields and their contents is presented to the user.

When the user searches in a database where some of the fields are hidden from him or her, a `Status` order will show only the fields he or she may read. The names of any hidden fields will not be recognized by the system if he or she uses them in a `Find`, `Show`, or `Define` order.

Not only is the user restricted from performing a search in the hidden fields by using their field names, but by default the results will contain no hits in the hidden fields when searching in the default `Vlew`, i.e. `TExt` and `PHrase` fields.

Although a user who is restricted from viewing certain fields will not know of their existence, he or she may be aware of lengthened response times for some searches. This time delay will be obvious, however, only if the search contains no target field in which to search, for example:

```
Find white rabbit ↵
```

### Hidden Fields and Output Formats

A user can never read the contents of hidden fields by giving a `CCL Show`, a `Print` or a `Print Local` order. The predefined reports are at his or her disposal, but they will output only the fields that he or she is allowed to see. This applies also to run-time definition of personal reports.

A database administrator should keep in mind that users may have limited read access to a database when designing reports.

Text inserts will always be output wherever they are positioned in the format, even if they pertain to a box containing hidden fields. The headers of hidden fields may thus appear, confusingly enough, in a format applied by a user with no read access to those fields. This can be avoided if such information is defined as headers of fields rather than text inserts.

### Hidden Fields and Data Entry

A user will be unable to delete records unless he or she has write privileges to the entire database. Fields for which the user has no write access will be blocked from data entry.

## Transferring Database Ownership

To transfer manager responsibilities from your own databases to another database administrator, select the database that you wish to transfer and select 'All Tasks...', then 'Change Manager' in the action menu:

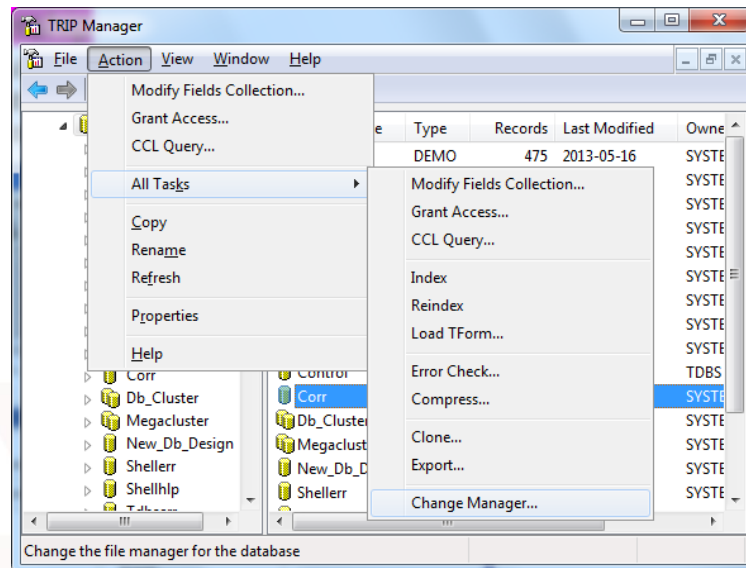


Figure 11–8 The Change Manager action menu option

A selection box will appear, allowing you to choose the new manager from a drop-down list:

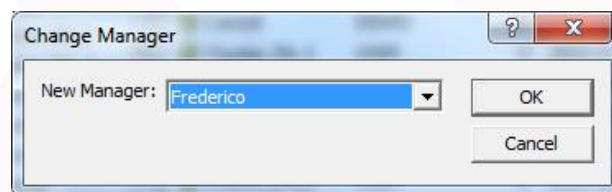


Figure 11–9 Change Manager Selection

Select the name of the new database administrator then click on the 'OK' button to confirm your choice. A pop-up confirmation will then appear.

The user SYSTEM may transfer the ownership of any user's database to anyone else using this form.

## Related CCL Commands

### Show

To display an overview of the access rights to your own databases, use the order:

Show ACcess ↵

or

Show BASe ACcess ↵

Databases are given alphabetically, while users and groups are listed chronologically within that database according to their user creation date. 'ALL', 'NONE' or 'SELECT' access information for read and write is provided for each user and user group.

To list the access privileges for an individual database, use

```
Show ACcess R=databasename ↵
```

which produces an output like this:

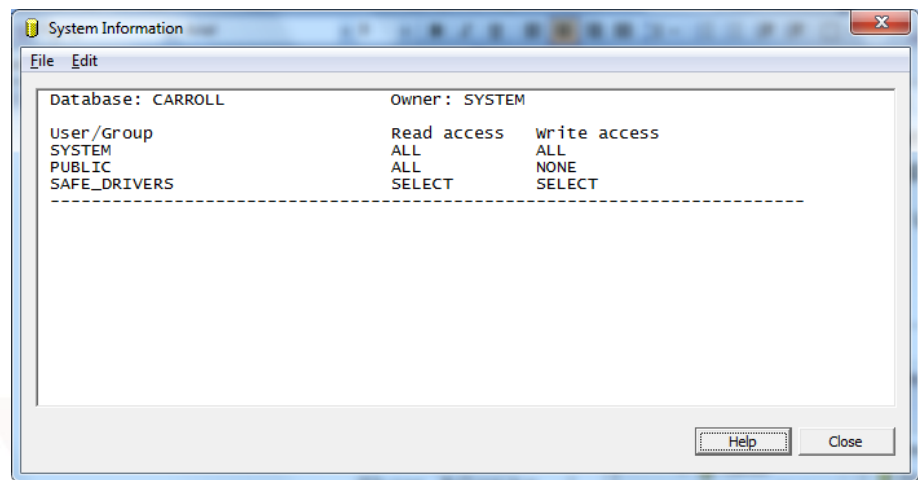


Figure 11–10 Carroll's Show ACcess screen

## Print

To send the listing to a printer or write it to a file, use the corresponding Print orders

```
Print ACcess ↵  
Print BAsE ACcess ↵  
Print BAsE ACcess File=filename ↵
```

and

```
Print ACcess R=databasename ↵  
Print ACcess R=databasename File=filename ↵
```

## Part 5:

## Appendix and Index



## Appendix A

### General Settings, Limits and Defaults

#### Support for the Euro Currency Symbol

If TRIPclassic is run using a terminal emulator, you will need to make sure that the emulator supports the Euro symbol and that it can define the host character set to be Windows Latin-1 (CP1252) or Windows EE (CP1250, Estonian).

*Note:*

- *If your host character set is incorrectly defined to be DEC supplemental or Latin-1 (ISO 8859-1), then some emulators will translate the Euro symbol from hex'80' to hex'A8' or hex'A4'. This value will then be transferred to TRIP, which will store it without any translation.*
- *The result is that the stored character will no longer be regarded as a Euro symbol when presented in a TRIPclient application, or when running the emulator with the correct Euro symbol settings. So it is important to ensure any terminal emulator is correctly set-up before entering the Euro symbol into TRIP via TRIPclassic.*

#### Searching for the Euro symbol

If the Euro symbol is to be a searchable character in TRIP then it must be defined as such in the database design. If it is not, then the Euro symbol will be ignored during indexing.

#### Support for the Chinese character set GBK.

TRIP supports the GBK character code. GBK includes more characters than GB-2312 (the standard Chinese character code in TRIP, defined as CHI) but the structure is similar. GBK is activated by setting the logical name TDBS\_CHARS to GBK.

Existing databases already indexed with CHI need re-indexing to upgrade to GBK.

#### Limit to TRIPclassic CCL Command Length

The maximum length of a CCL command in TRIPclassic is 400 characters. In applications created using the newer TRIPnxp and TRIPjxp APIs including TRIPmanger, there is no such limit.

*Notes:*

- *It is also possible to avoid the 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.*



## No Limits to Database and Index File Sizes

TRIP is perfectly capable of reading/writing database and index files of larger than 2GB, depending on the file system in use.

*Note:*

*If you are unsure as to the maximum single file size supported by your particular operating system, we recommend that you check in the file system documentation to ensure that files of larger than 2GB are, in fact, supported.*

## Limit to the Number of Search Sets

The theoretical maximum number of search sets in a single session is 65,536. However, certain system limits (e.g. available memory) may be exceeded before this limit is reached.

## Limit to the Number of Open Databases

The limit to the number of simultaneously open databases in TRIPsystem is 250.

*Notes:*

- 1 In TRIPclassic, the maximum length of any CCL order is 400 characters, hence any command may not exceed this length when opening many databases in TRIPclassic.  
*A workaround for the TRIPclassic 400 character limit in (1) above, is to use the DEfine command to define clusters of up to 30 databases, then to open these defined clusters in 'clusters of (again, up to 30) clusters'.*
- 2 Whether created using TRIPclassic, TRIPmanager, or via an API, the total number of databases in a 'cluster of clusters' must never exceed the 250 limit, otherwise any such oversized 'cluster of clusters' will be unusable.
- 3 Certain operating systems' limits may need to be adjusted: E.g. Maximum number of simultaneously open files limit.

*For example, keeping within prescribed limits:*

```
DEfine CLU1=DB1,DB2,DB3...
DEfine CLU2=DBa,DBb,DBc...
...
DEfine CLUx=DBi,DBii,DBiii
BAS ALLCLU=CLU1,CLU2, ... CLUx
```

## Defaults for the DEfine command

The default definitions for TRIP can be listed by starting the CCL command line in a newly started TRIP session issuing the CCL command:

```
DEfine ?
```

For ease of reference, the output from the command is shown below.

```
DEFINE
Highlight = All
```

```
No focus
No merge
No reverse
Hold
Save base
Tstamp update
No stop word
Display no orig
Display freq = merge
Find = no Fuzz
Page
FIND      max = No limit
+         max = No limit
DISPLAY   max = 1000
SORT      max = 1000
MAP        max = 1000
DELETE    max = No limit

AND = AND.E
MASK = '#: !&'
TIMEFORM = 1
CENTURY MIN = 1953
FUZZ = 75, 5, 2, 1
ABOUT = 50, No Highlight
VIEW = TExt, PHrase
```

**Note:**

*While the default settings for Display, Sort and Map are 1000, they can be set at any value up to "No Limit".*

### TRIPserver Crash Handling (Windows only)

TRIP is known for and has proven to be an extremely reliable, efficient and stable platform; nonetheless, as can happen in any large and complex software product, crashes can, albeit rarely, occur. For this reason, in the unlikely event of a crash, the following behaviour has been designed in to TRIPserver for Windows:

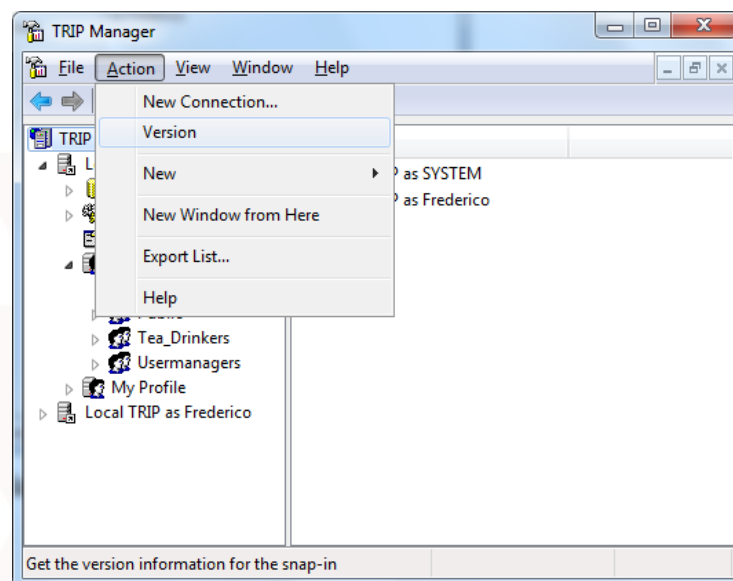
- Back-traces are dumped to file if it is the server (or any server-based utility/application) that is crashing
- Stack traces are saved in the TDBS\_LOG directory in files named backtrace\_nnn.log, where nnn is the process id that has crashed
- Any session is gently terminated, returning a message warning of a crash.

## Appendix B

### Obtaining Version and License Information

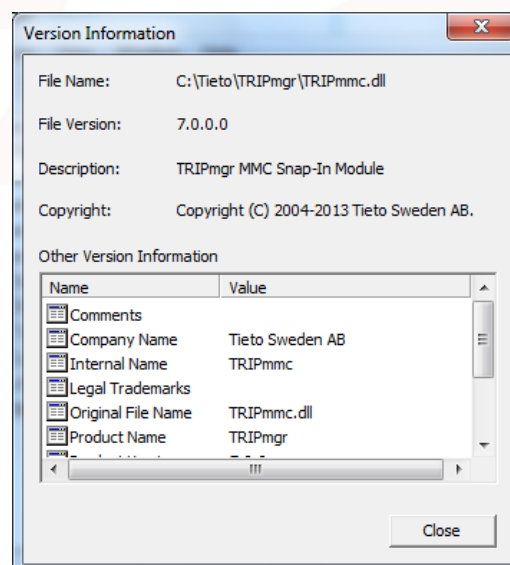
#### TRIPmanager mmc Version Information

In order to find the version information for the TRIPmanager mmc snap-in (for example when calling customer support), simply select the "TRIP Servers" node in the MMC tree, and choose the "Version" option from the "Action" menu:



**Figure B-1** Showing the mmc version

This will produce a dialog similar to Figure B-2 below:

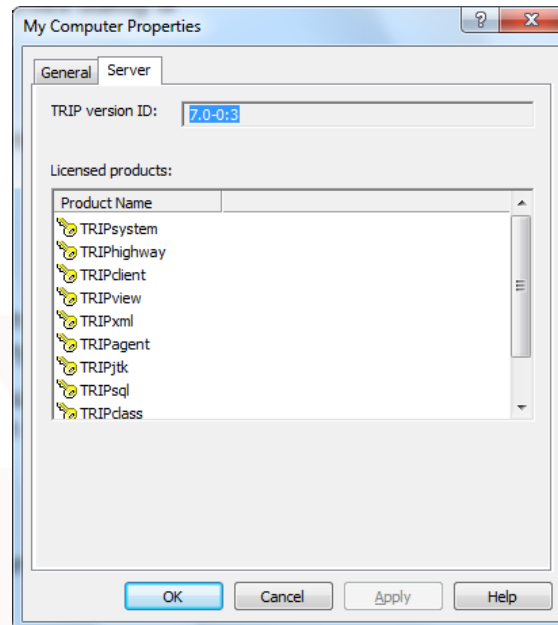


**Figure B-2** TRIPmanager mmc version information

Clicking on the 'Close' button will dismiss the above dialog.

## TRIPsystem Version Information

The TRIP system Version number can be ascertained by first selecting the TRIPserver node in question in the left-hand pane of the TRIPmanager mmc window, then selecting the 'Properties' option from the 'Action' menu. When the TRIPserver Properties dialog is displayed, select the 'Server' tab; the TRIPsystem version is listed in the 'TRIP version ID' box:



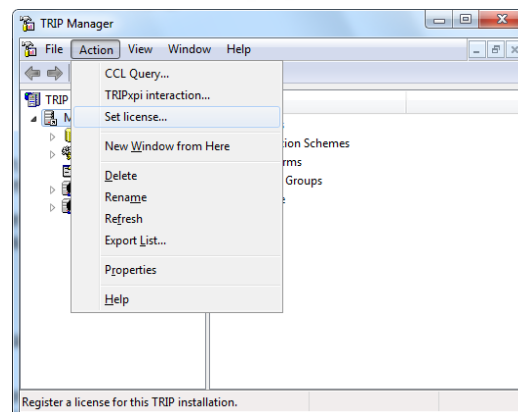
**Figure B-3 TRIPsystem server information**

## TRIP Product License Information

A list of those TRIP products currently licensed for this particular TRIP installation is below the TRIPsystem version information (See Figure B-3, above).

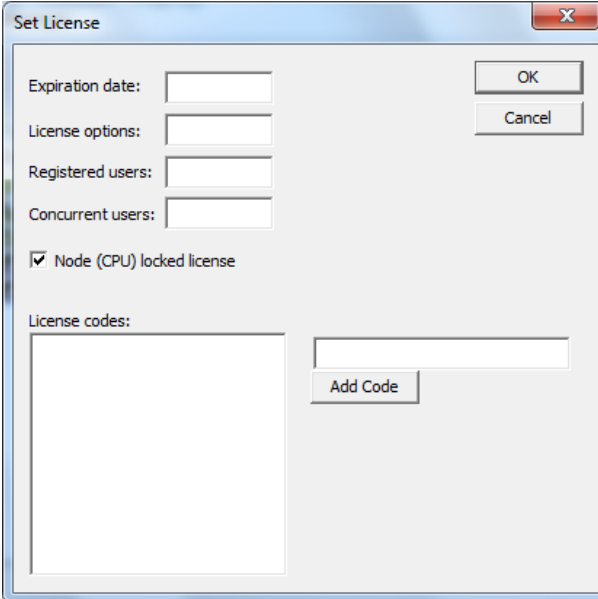
## Updating a TRIP Product License Key

Selecting the TRIPserver node in question in the left-hand pane of the TRIPmanager mmc window and then selecting the 'Set license...' option from the 'Action' menu:



**Figure B-4 Setting TRIPsystem license information**

will produce a dialog where it is possible to enter new TRIP license details (see overleaf):



The 'Set License' dialog box contains the following fields and controls:

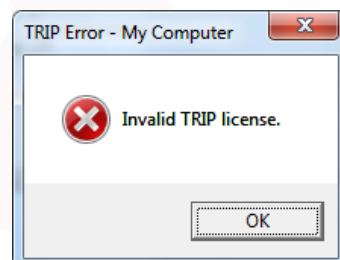
- Expiration date:
- License options:
- Registered users:
- Concurrent users:
- ☒ Node (CPU) locked license
- License codes: 

Add Code
- OK button
- Cancel button

**Figure B-5 TRIPsystem license entry dialog**

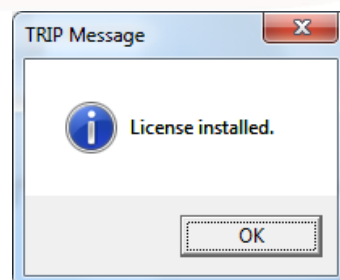
Enter the license key details from your 'packing slip' into the respective boxes and add the license code to the 'License Codes' list using the 'Add Code' button; when everything appears correct press 'OK'.

If incorrect details have been entered, an error message will appear:



**Figure B-6 Invalid TRIP license error**

If the entered details are OK, a confirmation message will appear.



**Figure B-7 License installed confirmation**

## TRIP User Account Validation Methods

### Overview

Part of the security of any computer system, or application, lies in controlling access to it from the 'outside world'; in this respect, TRIP is no different to any other application.

In order to control access, TRIP has three methods of user account validation, LDAP, Local System Validation and Standalone TRIP Usernames. Each is detailed in the following sections.

### LDAP

LDAP (Lightweight Directory Access Protocol) is an application protocol for querying and modifying directory services over TCP/IP.

Configuring TRIP login validation to use an LDAP repository, removes the need for users' passwords to be maintained in TRIP's CONTROL data dictionary.

However, in order to allow full control over access levels, a TRIP username must exist identical to the LDAP username, (See TDBS\_DISALLOW\_GUEST below, for more details).

*Notes:*

- *LDAP for TRIP is currently supported on the Windows, Linux and Solaris platforms*
- *The username SYSTEM is always validated against the local CONTROL database and is never subject to the directory service provider model (See 'TRIP Standalone Usernames', below).*

For detailed information on how to configure TRIP for LDAP authentication, refer to the document "TRIP LDAP Authentication" (UsingLDAPwithTRIP.pdf).

### Local System Validation

Local system validation (LSV) is a facility in TRIP for allowing automatic user validation for users already existing on the server hosting a particular TRIP installation.

Configuring TRIP login validation to use an LSV, removes the need for users' passwords to be maintained in TRIP's CONTROL data dictionary, thereby permitting a user to log into TRIP without entering a TRIP password.

For how to enable local system validation, see the 'Ignore Password' subsection of 'User Properties (1) – General', in Chapter 10 'User Privileges'

*Notes:*

- *The checkbox to ignore TRIP passwords also exists in the new user creation dialog (For more details see, "Creating a New TRIP User", also in Chapter 10).*
- *For this form of validation to work, the TRIP installation must be on the same server that is carrying out the validation; it is, therefore, only really of use in TRIPclassic sessions, or server based applications.*

## TRIP Standalone Usernames

The 'traditional' way of handling TRIP users, TRIP Standalone users are unique to each TRIP installation and are maintained in that installation's CONTROL data dictionary.

This method of user management does not represent any significant difficulties, other than the TRIP usernames and passwords may be different to those needed to access the operating system TRIP is installed on.



## Connecting to TRIP Servers

### Server Connection Overview

The fundamental requirement of any management activity is to be connected to a TRIP instance. This can be either local or remote, with the following rules:

- A given console can contain only one local connection. Once a local connection has been established, the option to establish a local connection is no longer available in the connection wizard.

*Note:*

*It is possible to construct two or more local connections by using two or more consoles within the same MMC process (i.e. by creating two or more .MSC files). However, this behaviour is not supported and may lead to unpredictable results.*

- A given console can contain as many remote connections as desired, each of which may be targeting a different TRIP server, or the same TRIP server with a different set of login credentials or host initialization script.
- A connection, either local or remote, is identified by its "alias." This alias is a string of arbitrary length that should be meaningful to you as a means of identifying the connection, without having to examine the server's properties.

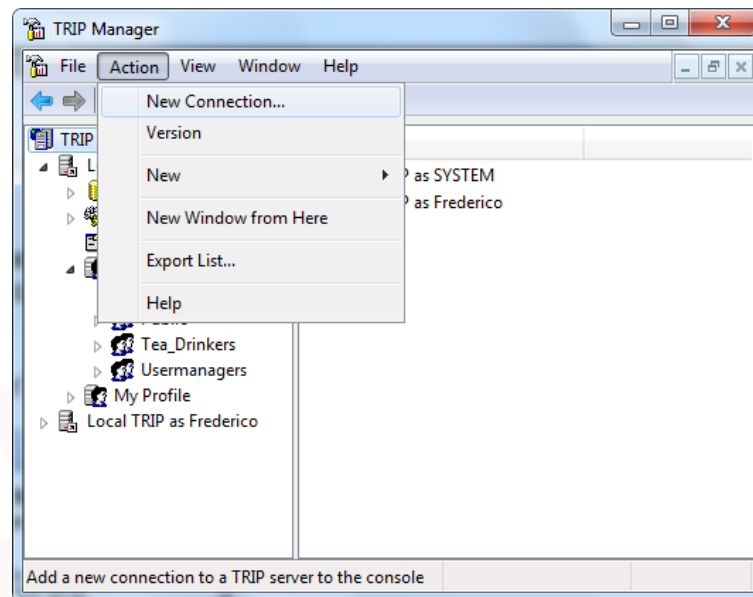
*Note:*

*The connection alias does not have to be unique, although it will no doubt be confusing if it is not.*



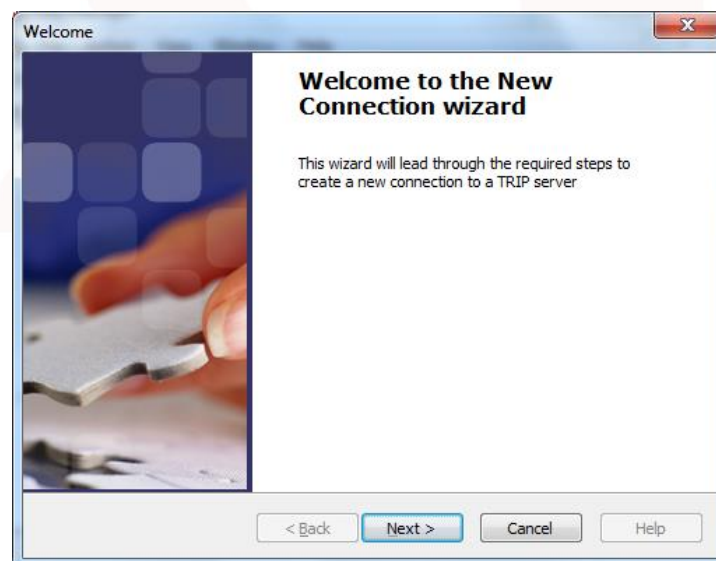
## Creating a Server Connection

In order to connect to a TRIP instance, simply select the "TRIP Servers" node in the MMC tree and choose the "New Connection..." option from the 'Action' menu:



**Figure B-8 Selecting the New Connection Wizard**

You will be presented with the "New Connection" wizard, as shown overleaf:

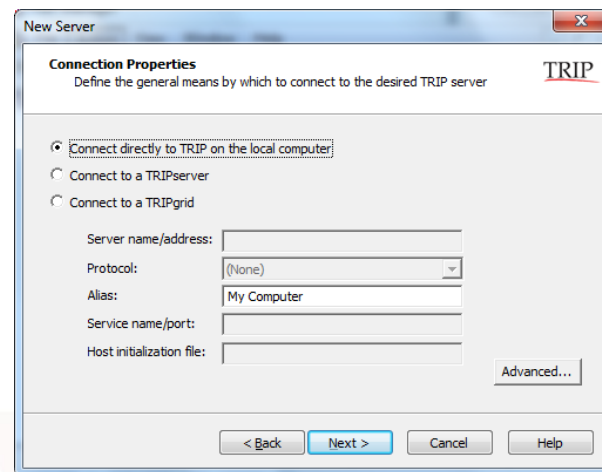


**Figure B-9 The New Connection Wizard welcome dialog**

Clicking the "Next" button will progress to specifying the connection type and parameters; see overleaf:

## Local Connection

As shown below in Figure 2.37, when connecting to the local TRIP instance, no further information is required:



**Figure B-10 Specifying a local Connection**

The default alias is "My Computer" but you can change this as desired.

*Note:*

*It is only possible to create one local connection in TRIP manager. If you need to create more connections to the local machine, use Remote Connections and specify 'localhost' as the machine address.*

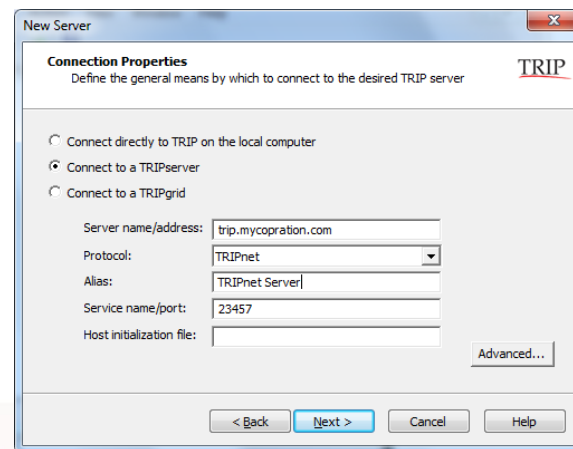
## Remote Connections

In order to construct a remote connection using TRIPnet, you need to know the following information:

- The name or IP address of the machine on which TRIP is installed
- The protocol by which the connection will be made:
  - TRIPnet
  - Encrypted TRIPnet
  - XML over HTTP (via a Web proxy)
- The alias by which this connection will be identified within the TRIPmanager mmc console (this default's to the machine's name or IP address)
- The port number or service name that uniquely identifies the TRIP server on the target machine (the default is use port number 23457)

- The name of an (optional) host initialization script that the server should run to configure its operating environment.

For example:



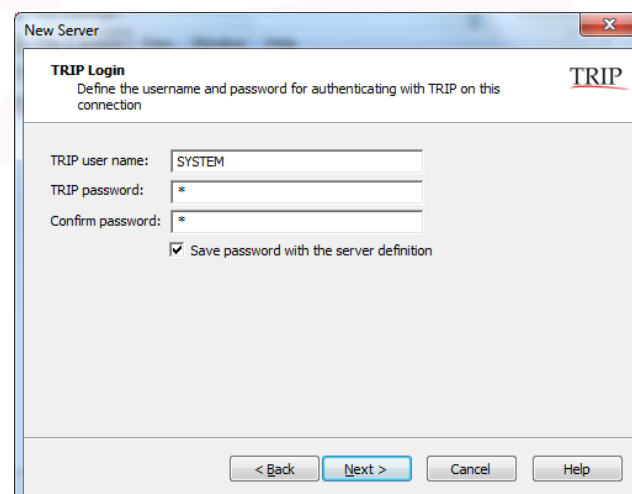
**Figure B-11 Specifying a remote connection**

Clicking on the "Next" button will proceed to specifying the login credentials to associate with this connection

## Specifying Credentials

This dialog allows you to specify the user name and password with which you wish to login to TRIP. The default behaviour is to not store the password within the console (the .MSC file) because that storage is not encrypted.

However, if you click the "Save password with the server definition" check box, you will be able to type in the password (and confirmation string). This password will then be stored along with the server definition in the console file and will be used every time the console attempts to login to that particular TRIP instance.

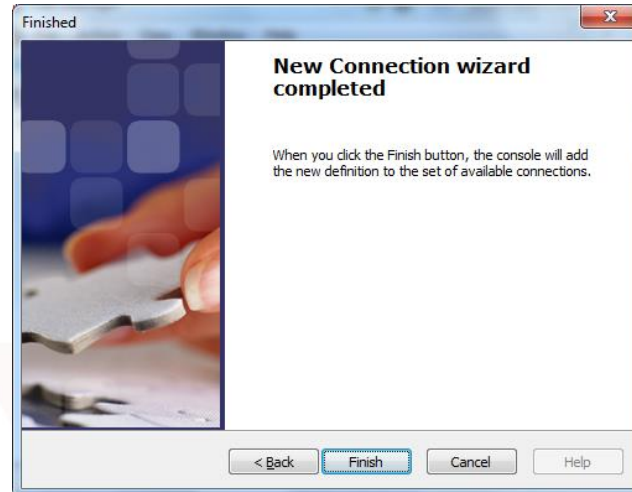


**Figure B-12 Specifying user credentials**

If you opt not to store the password with the server definition, every time the console needs to login to this TRIP instance, you will be prompted to provide the password.

If you change your mind later and wish to either store the password or to remove / change a stored password, you can use the Server's properties dialog to do so.

When you click the "Next" button, you will see a confirmation dialog that allows you a final opportunity to cancel this action:



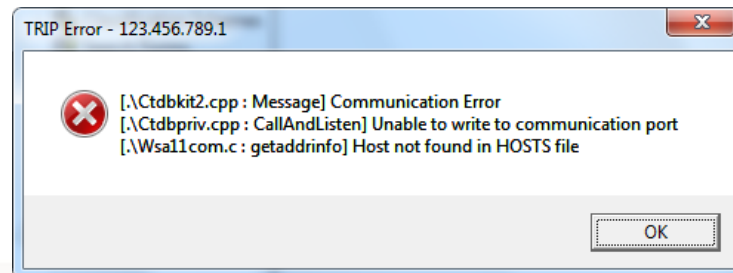
**Figure B-13 New Connection completion dialog**

Clicking "Finish" at this point will result in the new server's connection information being added to the console.

## Logging into the New Server Connection

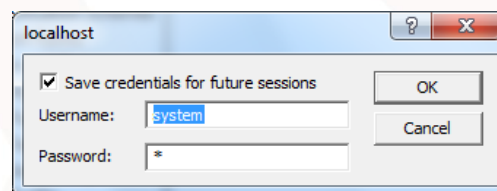
In order to actually login to the TRIP instance, click either on the new server node in the MMC tree, or on the plus sign (+) that appears to the immediate left of the server's alias. When you do so, you will see one of three things:

- An error dialog if, for example, the TRIP license is invalid for this instance, the password is incorrect, or the specified remote server could not be contacted:



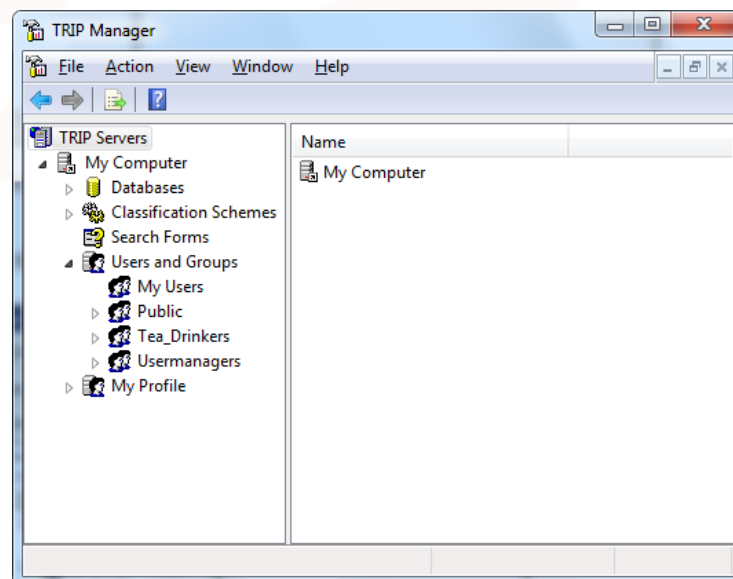
**Figure B-14 Communications Error**

- A login dialog, if you either didn't store the TRIP password or specified an incorrect password (to correct an invalid password, use the Server's properties dialog).



**Figure B-15 Login Dialog**

- A set of descendant nodes that specify the types of information that can be administered with this snap-in, signifying a successful login:



**Figure B-16 Successful login**

## TRIP Grids

Notes:

- *The TRIPgrid software is included as part of TRIPwpi*
- *The TRIP license installed on the TRIP server used for metadata and authentication must include TRIPgrid and TRIPjxp. The latter is because TRIPgrid uses TRIPjxp to communicate with the metadata database.*

### Introduction to TRIP grid computing

As data and user volumes increase it can quickly become difficult to manage either one or both of these metrics within the confines of a single server.

Rather than continue to throw more and more expensive hardware at this problem, TRIP grid computing allows for the construction of cheap commodity matrices of hardware that, in combination, can provide much greater throughput for both data and user volumes than would ever be possible within a single machine.

The core concept behind a TRIP grid is, by splitting a query into multiple parts (grids are really intended for use in read-often / write-rarely configurations), each of which can be serviced by a different server, the aggregate throughput of the whole grid will be considerably higher than would be possible otherwise.

To achieve this, TRIP grids support two key notions, one being clusters, the other being replica sets:

- A replica set is a set of databases on one or more physical servers, each of which is considered a duplicate (or replica) of the others. There is nothing explicit within the grid logic that enforces this; it is entirely up to the grid administrator to create the replica relationship using TRIP's normal log file-based roll-forward replication mechanisms.
- A cluster is a collection of either physical databases or replica sets on one or more servers that are to be searched together, much like a cluster definition on a single server. The cluster is the primary searchable entity within a TRIP grid.

Queries placed against a grid are, in fact, placed against a cluster hosted by that grid.

Note:

*All current programming interfaces support queries against grids as well as against physical servers, i.e. TRIPnxp, TRIPjxp, TRIPaxp)*

The grid router (a web service hosted on one or more of the servers taking part in the grid) is responsible for breaking the grid query into as many parts as are necessary, in order to dispatch the query to all physical grid machines taking part in that cluster.

Note:

*Databases within a replica set are used in a 'round robin' fashion to attempt to load balance user volumes against the available data.*

As an example, consider a grid consisting of three machines:

- *grid\_1* hosts the grid router
- *grid\_2* hosts db1 and db2
- *grid\_3* hosts db2 and db3

Now assume that we construct the following logical entities within the grid:

- *rep\_1* is a replica set consisting of *grid\_2.db2* and *grid\_3.db2*
- *cluster\_1* is a cluster consisting of *grid\_2.db1*, *grid\_3.db3* and *rep\_1*

Queries placed against this grid, using *cluster\_1* as the search domain, will therefore always be dispatched to at least two query servers by the grid router. For example, a simple search ("Find 'x'") against *cluster\_1* will result in two queries being dispatched:

- *grid\_2* is told to query *db1* and possibly also *db2*, depending on the replica set load balancing
- *grid\_3* is told to query *db3* and possibly also *db2*, depending on the replica set load balancing (obviously, only one of *grid\_2* or *grid\_3* would be directed to query against *db2*)

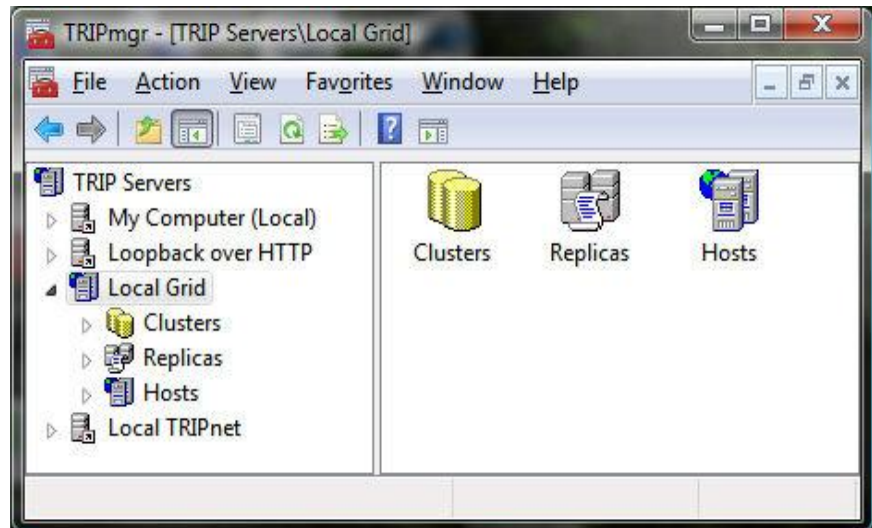
The grid router is then responsible for collecting the results from *grid\_2* and *grid\_3* and collating them prior to dispatch to the query originator. This collation could be caused by sorting on one or more key fields, sorting by ranking, or a combination of both.

In absence of specific collation criteria, the final result set will be a round robin collation produced by taking the *first record in the search results (RIS 1)* from *server 1*, then *RIS 1* from *server 2*, ..., then *RIS 2* from *server 1*, etc.

Constructing a TRIP grid is therefore an exercise in deciding which type of scalability you most wish to emphasise:

- For more data, partition the data across multiple machines using a grid cluster: e.g. by splitting an existing database cluster
- For more users, replicate high throughput databases across multiple machines using a replica set

Grids are represented within TRIPmanager as servers, but with a different set of descendant nodes than a normal physical server connection, for example:



**Figure B-17 A TRIP Grid in TRIPmanager**

As the names imply, the clusters and replicas nodes allow for management of clusters and replica sets, respectively. The hosts node simply allows for navigation and inspection of which databases are being served by which physical machines.

*Note:*

*In order for a machine to take part in a TRIP grid, it must be a TRIPnet server as this is primary means of communication between the grid router and the grid members.*

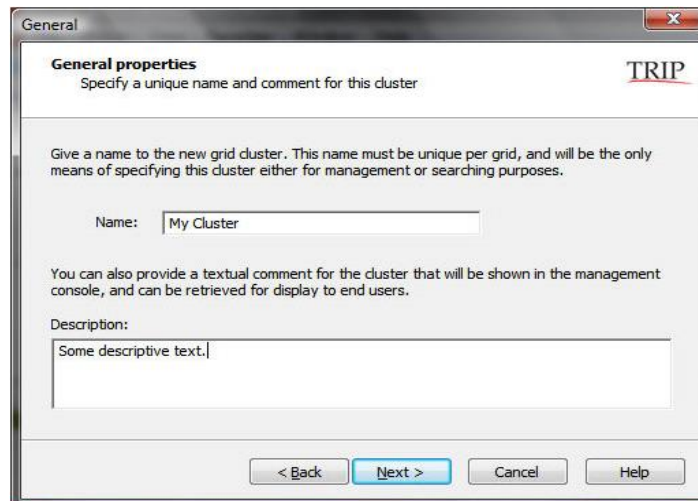
## Creating a Grid

Creating a TRIP Grid requires installation and configuration of software that is beyond the scope of this document. For more information, see the document entitled, "TRIPGRID USER'S GUIDE", which is included in the TRIPwpi distribution.

## Creating a Grid Cluster

To create a new cluster, simply right click on the "Clusters" node and choose "New Cluster..." This will produce a wizard that leads the user through providing the required information, which is basically just the name and description of the cluster. The name must be unique across the grid, but has no other requirements.





**Figure B-18 Defining a cluster**

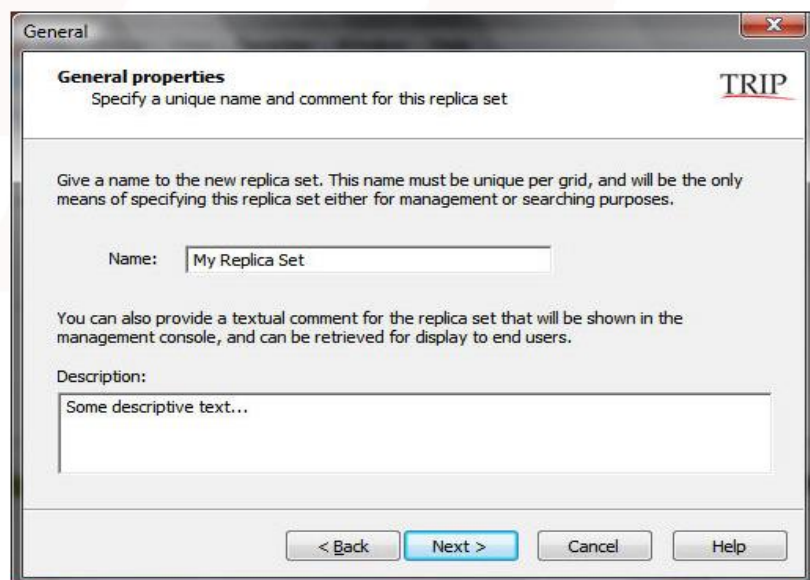
Once the cluster is created, add databases and replica sets to it using the publication mechanisms described in the following sections overleaf.

### Creating a Grid Replica Set

To create a new replica set, simply right click on the "Replicas" node and choose "New Replica Set..." This will produce a wizard that leads the user through providing the required information, which is basically just the name and description of the replica set.

*Note:*

*The name must be unique across the grid, but has no other requirements.*



**Figure B-19 Defining a replica set**

Once the replica set is created, add databases to it using the publication mechanisms described in the following sections overleaf.

## Publishing to a Replica Set

In order to publish one or more databases to a replica set, simply find those databases in TRIPmanager and drag / drop them to the replica set. That is, the server that is hosting the physical database to be added must already be configured for access via TRIPmanager, and must be present within the console definition file in use.

The connection configured must be via TRIPnet as no other communication mechanisms are supported for use between the grid router and grid member servers.

If the server hosting the database being added to the replica set is not currently a member of the grid in question, you will be prompted to confirm that you wish to add this server to the grid.

## Publishing to a Grid Cluster

In order to publish one or more databases to a cluster, simply find those databases in TRIPmanager and drag / drop them to the cluster. That is, the server that is hosting the physical databases to be added must already be configured for access via TRIPmanager, and must be present within the console definition file in use. The connection configured must be via TRIPnet as no other communication mechanisms are supported for use between the grid router and grid member servers.

If the server hosting the database that is to be added to the cluster is not currently a member of the grid in question, you will be prompted to confirm that you wish to add this server to the grid.

You can also add replica sets to a cluster; in order to do so simply drag the appropriate replica set to the cluster definition.

## Grid Authentication

In contrast to normal TRIP connections, grid connections are not persistent. This means that each query operation creates its own connections to as many TRIP servers as are required to service the query and once the query is complete, those sessions are terminated. This keeps the grid clean in terms of processes in use, but it does place a burden in terms of how sessions are authenticated, and how many sessions are continually being created and terminated.

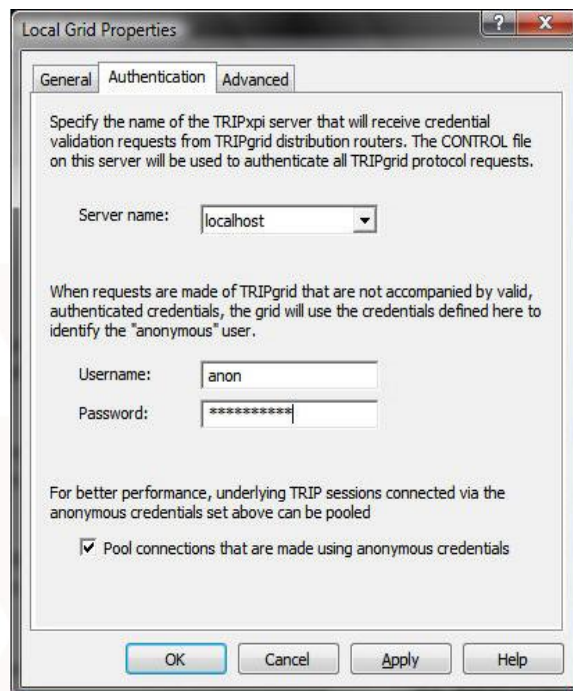
In order to use the grid in authenticated form, i.e. using discrete credentials for each user of the grid, each server that is taking part in the grid must have a coherent copy of some central CONTROL database containing user credentials and access rights for each potential user. This allows for each query to be accompanied by authentication information (i.e. user name and password) and for that to be forwarded to each server taking part in the query. In turn each server will use that authentication information when creating the TRIP session for the query operation.

Whilst this provides a high degree of control, it does also place a high burden of replication upon the grid administrator.

To provide an easier to administer mode of operation, and to reproduce a much more "real life" interaction model, TRIP grids also support anonymous and pooled access.

What this means is, if a query is not accompanied by authentication information, that query is performed within the context of a predefined anonymous user. As a secondary benefit, anonymous user connections can be pooled per server, so as to use the minimum server resources possible.

In order to establish anonymous credentials for a grid and to enable connection pooling for those anonymous sessions, simply provide credentials on the Authentication tab of the grid's property sheet (see overleaf):



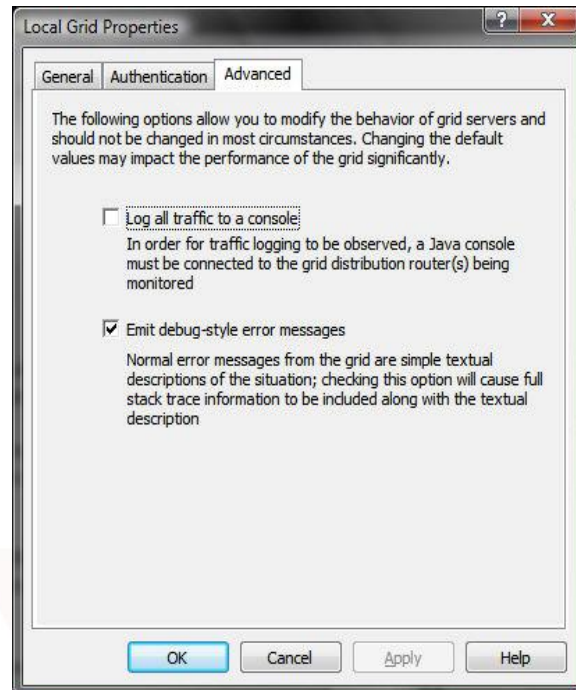
**Figure B-20 Grid Authentication tab**

*Note:*

*The server defined as hosting the authentication CONTROL database must already be a member of the cluster, i.e. it must be publishing at least one database to a cluster somewhere.*

### Advanced Grid Properties

This property page displays several advanced properties that are really meant for debugging grids that aren't working correctly:



**Figure B-21 Advanced Grid Properties tab**

The first option, to log all traffic to a console, requires a Java console to be attached to the grid router; for example, *Eclipse* using the *Sysdeo Tomcat* addon.

The second option, to emit debug-style error messages, is probably much more use in a customer situation. In essence this directs the grid router to emit error messages complete with a trace of where the error occurred, so that this can be forwarded to TRIP support for triage and follow-up.

## Classification Schemes

Note:

*The logical name `TDBS_Fel`! Hittar inte referenskölla. must be configured in the [Non-privileged] section your `tdbs.conf` file for classification schemes to function correctly: For more information, see page **Fel! Bokmärket är inte definierat.** of this document.*

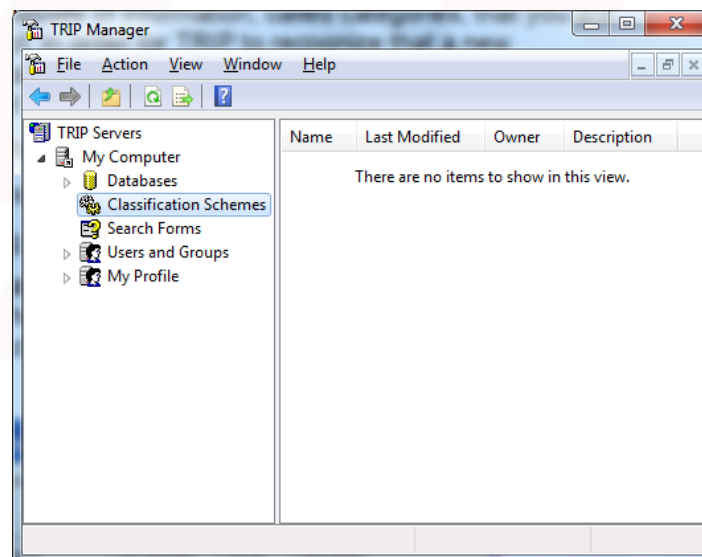
### Introduction to Classification Schemes

A classification scheme is a collection of information (in reality, a special-purpose TRIP database) that instructs TRIP on how to recognize documents as representing one or more classes of information. The classes of information, called categories, that you are interested in are defined by you and, in order for TRIP to recognize that a new document belongs to a particular category, you must train TRIP using documents that you know are representative of that category.

The classification process is therefore divided into two steps:

- Management, or training and definition
- Categorization, or assigning categories to documents being indexed

You can accomplish everything related to classification scheme management under the "Classification Schemes" node in the MMC tree. The list of objects that you see as sub-nodes in the MMC tree are those schemes to which you have at least some level of read access.



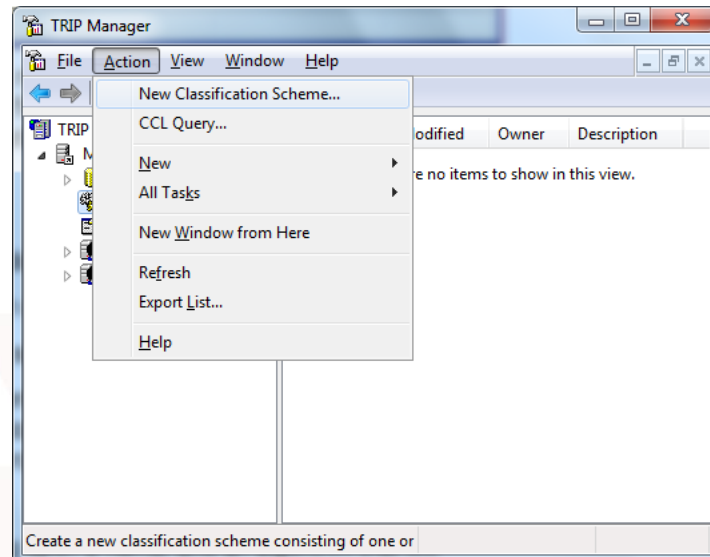
**Figure B-22 Classification Schemes sub-node**

For example, in your business you may determine that it's important to be able to correctly separate documents of a financial nature from those of a legal or product nature. Creating a simple classification scheme that recognizes these three categories of documents is accomplished by:

- Creating a new scheme
- Creating the categories of document that you wish TRIP to be able to recognize

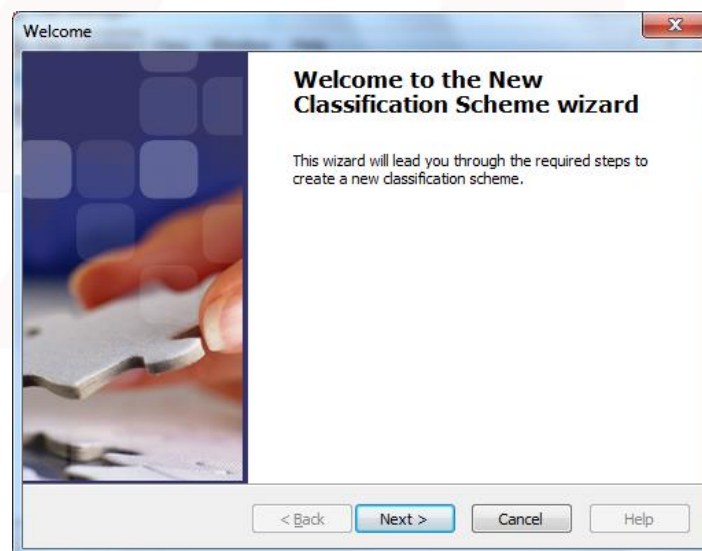
- Training each category with documents that represent the category
- Attaching the classification scheme to one or more databases

To create a new classification scheme, simply right click on the "Classification Schemes" node and choose "New Classification Scheme..." from the 'Action' menu.



**Figure B-23 New Classification Scheme menu item**

This will produce a wizard that leads you through the process of creating a new scheme:



**Figure B-24 Create New Classification Wizard**

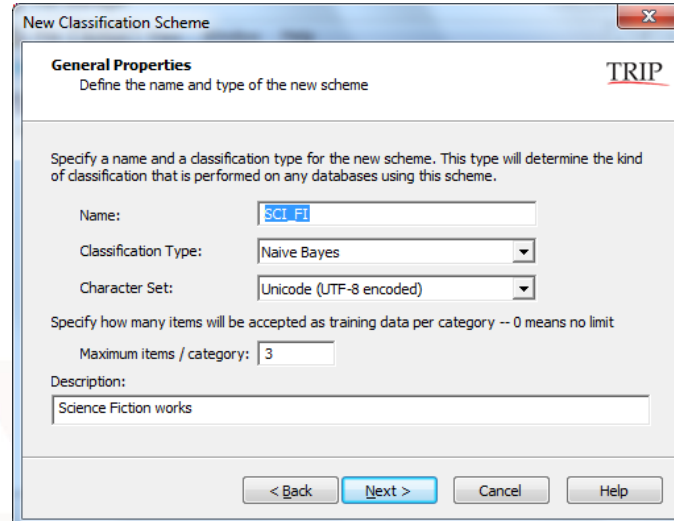
Configuring the new scheme is simply a matter of:

- entering a name for the new scheme
- selecting a classification algorithm for training categories and assigning tags
- choosing a character set to be used for the classification scheme's storage database



- setting the maximum number of categories/items to be accepted as training data
- entering a description

as shown on the next page:

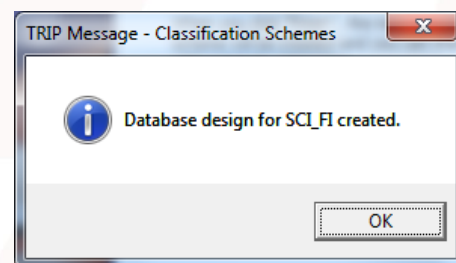


**Figure B-25 New Classification Scheme properties**

*Note:*

*Currently, the only classification algorithm available is 'Naïve Bayes'.*

Successful creation of the new classification scheme will be indicated by a TRIP message:



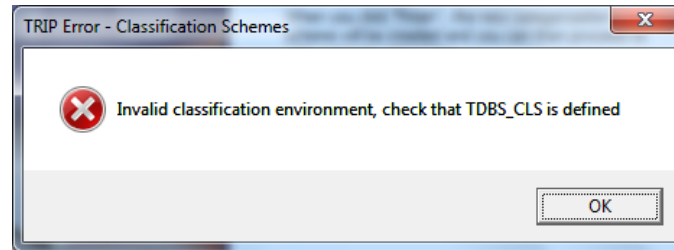
**Figure B-26 Classification Scheme storage database created**

### Attaching a Classification Scheme to a database

To attach a classification scheme to a database, in order that the records in that database are categorized whenever new or updated records are processed for indexing, use the Database General Properties dialog, as described beginning on page 31 of this guide.

*Note:*

*If the TDBS\_Fel! Hittar inte referenskölla. logical name has not been correctly configured in the [Non-privileged] section your tdb.conf file, the following error will appear when you try to submit your new scheme:*



**Figure B-27 TDBS\_CLS configuration error**

*For more information on configuring TDBS\_Fel! Hittar inte referenskälla., see page **Fel! Bokmärket är inte definierat.** of this document.*



## Scope Search Facility

*Note:*

*As has been stated elsewhere in this manual, the “UPDate SCoPe” command is not connected with global updating. Aside from this Appendix, further information can be found in the “Find SCoPe” and “UPDate SCoPe” sections of the CCL Command Reference.*

### The new Scope Search facility

UPDate and Find SCoPe functions are used, respectively, to update and search using predefined saved search sets, in order to be used across large database clusters of mostly static data. For example, such a cluster might contain historical data split across several databases, one for each year; the most recent database (i.e. the one for the current year) being the only one that has data that changes and is still being updated.

In such a large database cluster, it may be useful to have several TRIP procedures – e.g. one each for particular 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 only for searching by TRIP, in order to simplify, standardize and speed up such searches.

### Scope Search Example

This example is set in 2011 and uses a database cluster of eleven TRIP databases.

The first part of the cluster comprises one database for each of the years from 2001 to 2010, named *db01* to *db10* respectively, and they contain the historical data. As the data these databases is essentially static, they are never updated and only ever used in searches.

There is also one extra database, *db11*, for the ‘current’ year (2011) which is updated throughout the year as new data is added.

There are also three TRIP procedures, each one for different areas of interest and resulting in a different search set. These procedures, named *proc1*, *proc2* and *proc3* are for creating pre-made search sets that can be used in order to simplify, standardize and speed up the searching.

*Note:*

*In the following example, the number of search hits, usually displayed in the search history as an integer, is represented by <N1>, <N2>, etc.*

### Setting Up the Scope Search

To set up the Scope Search facility, do the following:

- Start TRIP
- Open the databases that together define the cluster:  

```
base DBCL=db01,db02,db03,db04,db05,db06,  
db07,db08,db09,db10,db11
```

  
resulting in `S=1`     `<N1>`     `base DBCL=db01,db02, ... ,db11`
- run all TRIP procedures that create pre-made search sets as follows:

```
scope(proc1) resulting in S=2      <N2>      scope(proc1)
scope(proc2) resulting in S=3      <N3>      scope(proc2)
scope(proc3) resulting in S=4      <N4>      scope(proc3)
del s=1 to remove the cluster creation command
```

- save the above search sets in a special SIF file that will be used by TRIP for searching:

```
stop save no highlight file=special.SIF
```

**Note:**

Any name can be used for `special.SIF` and specifying “no highlight” will keep the size of the SIF file down and thus help speed up the searches; however there will of course be any highlighting of the pre-searched terms.

To get highlighting use the following command:

```
stop save file=special.SIF
```

- a logical name pointing to the special SIF file should be defined in the environment for each user:

```
TDBS_PRE_SCOPE=/path-to-SIF-file/special.SIF
```

## Using the Scope Search

To use the new Scope Search facility, do the following:

- start TRIP (with TDBS\_PRE\_SCOPE set as described above)
- open the same cluster as when the special SIF-file was created, with the databases in the same order:

```
base DBCL=db01,db02,db03,db04,db05,db06,
      db07,db08,db09,db10,db11
```

resulting, as before, in

```
S=1      <N1>      base DBCL=db01,db02, ... ,db11
```

- perform a search thus:

```
find scope(proc2) and (any other search criteria)
resulting in
```

```
S=2      <N2>
```

- this search should use the pre-made search saved in the file pointed to by TDBS\_PRE\_SCOPE and this should be faster than performing the search without the pre-made search sets.

## Updating the Scope Search

When the database for the current year is updated, or if a change is made to one of the static yearly databases, the special SIF file must be updated.

To update one database in the special SIF file:

- make a backup copy of the special SIF file.
- copy the special SIF file to 'username'.SIF, e.g. as user SYSTEM:

in UNIX:

```
cp /path-to-special-SIF-file/special.SIF SYSTEM.SIF
```

in windows you can simply copy/paste then rename the file

- start TRIP (with TRIP's home directory set to where SYSTEM.SIF is located)
- the search sets created above will now appear, in this case:

```
S=1      <N1>      scope (proc1)
```

```
S=2      <N2>      scope (proc2)
```

```
S=3      <N3>      scope (proc3)
```

- update (for example) the db05 database in these search sets, thus:

```
upd scope (db05)
```

- this should update the search sets with the result of new searches for db05; no new search sets are created:

```
S=1      <N4>      scope (proc1)
```

```
S=2      <N5>      scope (proc2)
```

```
S=3      <N6>      scope (proc3)
```

- save the search sets in the same way as before, using the same file name:

```
stop save no highlight file=special.SIF
```

or

```
stop save file=special.SIF
```

- an updated version of the special SIF file will now exist, and users who are using it at this moment, will immediately get access to the updated file.

*Note:*

*When adding a database for a new year, the special SIF file will have to be created. This is done exactly as creating a SIF file is described above, but adding the new database to the list of databases that make up the cluster.*

## Appendix C:

### TRIP Programming

This part of the appendix contains information valuable to programmers who will be responsible for writing:

- applications to convert online data to TForm
- add-on modules to give TRIP more functionality using ASEs

*Note:*

*For more detailed descriptions and examples on use of the TRIP Application Programming Interface (API), refer to the TRIPsystem API Reference Guide provided with the TRIPsystem release documentation.*

### TForm

The TRIP system offers two main methods for entering data into a database:

- manual data entry, and
- automated loading of machine-readable data by conversion to TRIP's input format TForm, and entry into the BAF using the LOAD procedure.

TForm is a delimiter-controlled record format for the transfer of text into records intended for a TRIP database. Using TForm, sequential text files (variable length record format) using the DEC multinational character set or in 7-bit ASCII may be entered into a TRIP BAF file.

A BAF file consists of a sequence of records, each record containing one or more fields, and a field consists of one or more subfields or paragraphs. The paragraphs are further subdivided into sentences.

A TForm file is a text file with control strings, which determine how the text strings will be organized in the BAF. These control strings adapt the file contents to the structure of the database by marking the beginning of the individual record (and record part), the beginning of the individual field, and its subdivision.

### Control Strings

The control string delimiter generally used is the caret [^], followed by an alphanumeric marker. The following characters identify the five basic types of markers:

Marker Type	Symbol
Record	R
Record Part	G
Field	F
Paragraph/Subfield	P
Sentence	S

A control string may also contain control skip characters, which allow the insertion of spaces and linefeeds for ease of proofreading and editing of TForm files. These control skips are ignored when the file is transferred to a BAF file.

All characters with a decimal ASCII representation of up to and including 32 (<SP>), or any combination of these, will be accepted as control skip characters. A string of appropriate characters may immediately follow a delimiter or record marker, but not a field, subfield or sentence marker.

*Note:*

*When the content of a field is strictly regulated, as in the case of NUmber, INteger, DAte, TIme or PHrase fields with a pattern, you should place the defined delimiter immediately after the subfield content to avoid including extra characters (space or <Return>) in the field contents.*

When a TForm file is transferred to a TRIP file, the following situations hold true:

- ^R<CR><LF>^F is a record control string [^R] followed by a field control string [^F], and is equivalent to ^R^F,
- however, ^F<CR><LF>^P is a field control string [^F], followed by the text string <CR><LF> and a subfield or paragraph control string [^P], and is not equivalent to ^F^P.

A control master is available to support the available character sets. For example,

^CROM

inserted at the beginning of a TForm file tells TRIP that this file is written in character set 'Roman 8'.

This delimiter can be defined differently for each TForm file. The first character in a TForm file tells TRIP what the delimiter is going to be.

## Text Strings

A text string is a sequence of characters bounded to the left by a field or subfield control string, and to the right by a single delimiter (or the end of the TForm file).

Normally, TRIP determines automatically what constitutes a sentence or paragraph in a TExt field. Should you wish to define sentences manually, you must use a sentence or paragraph marker before every sentence and paragraph in the text portions of the records.

When the TForm file is loaded into TRIP, the contents of TExt fields are kept in their original form, unless the database manager has decided otherwise during design. 'Layout retained' ensures that linefeeds and blanks are kept exactly as they are in the original. The one exception to this is the blank line marking the start of a new paragraph.

## Record, Record Part, Field and Subfield Markers

The six basic markers used to create a record are record, record name, record part, field, paragraph/subfield and sentence markers.

### The Record Marker: nR

The marker R signals that record n is to follow (n is an integer), or, if n is omitted, that a new record is to be added at the end of the BAF. The record number is used only to identify an already existing record when updating it.

The record marker must be immediately followed by a new control string, or by a string of control skip characters followed by a control string. One exception occurs when using a record name while updating.

### The Record Name Marker: N

N (followed by a record name) signals that a record with the given name is to be added, or, if a record by that name exists already, that it is to be updated.

### The Record Part Marker: nG

Marker G indicates that record part n is to follow (n is an integer), or, if n is omitted, that a new record part is to be added at the end of the record. The record part number is used only to identify a previously existing record part when updating it.

A record with record parts in a TForm file should start with the head fields, followed by the part fields of each record part.

### The Field Marker: nF

This marker directs that a field n of the current record is to follow (where n is an integer).

### The Paragraph/Subfield Marker: nP

P signals that paragraph/subfield n is to follow (n is an integer), or, if n is omitted, that a new subfield or paragraph is to follow at the end of the current field.

In a TExt field, TRIP recognizes a new paragraph as the end of a sentence, followed by two <CR><LF>s and the start of a new sentence. This is the system default both at data entry and in a TForm file. P as a paragraph marker is redundant if paragraphs are separated in this manner.

*Note:*

*If paragraph and sentence markers are used, data entry forms must not be used for these records.*

Any given text string will be assigned to the subfield given by the control string preceding it. If this is a field control string, then the text string is assigned to a new subfield at the end of the indicated field. This makes ^2Ftext string^ and ^2F^Ptext string^ equivalent, and if field two in the current record is a new field, then both are equivalent to ^2F^1Ptext string^. In that case, all three will put the string 'text string' in the first subfield of the second field of the current record.

### The Sentence Marker: S

The sentence marker may be useful if the text strings contain data that should not be interpreted as sentences. The default sentence definition is an end-of-sentence marker followed by at least one space and a capital or upper-case letter. TRIP would read such a sequence as the end of one sentence and the beginning of another if the string was controlled by a paragraph marker only.

## Adding Records with TForm

We will use the TRIP demonstration databases Corr and Carroll to describe how a TForm file is made. We will examine Corr first, which is structured as follows:

Field name	Type	No	Contains	
rname	PHrase	1	recipient:	name
rcomp	"	2	"	company
raddr	"	3	"	address
rcountry	"	4	"	country
sname	"	5	sender:	name
scomp	"	6	"	company
saddr	"	7	"	address
scountry	"	8	"	country
day	DAte	9	the date of the message	
cat	PHrase	10	type of communication	
content	TExt	11	the text of the message	

Each field in Corr is of one of the seven existing data types. Paragraphs and sentences are used in fields of the type TExt, while subfields are used in fields of the other six types (PHrase, NUmber, INteger, DAte, Tlme and STring).

Assume that a file of correspondence (letters and telexes) is to be entered into the Corr database. The same TForm layout is used for both initial record loading and for appending records to already existing data.

When you create a database, the system numbers the fields as you identify them. A SStatus or Show database order will display the database field numbers, presenting the fields in field number order. TForm files present the only occasion where you will use field numbers instead of field names.

When designing a database, the database manager decides whether a TExt or PHrase field is to keep its original layout ('Layout retained'). Here, all <Tab>s, <LF>s, and spaces are maintained as they occur in the entered text, whether the data has been imported from a TForm file or has been entered manually during data entry.

As these records were loaded into TRIP in their original form, there is one empty line before the first paragraph in field eleven. A sentence separator [. ! ?] followed by two <CR><LF>s and the start of a new sentence marks a new paragraph by default, so no paragraph or sentence markers are needed.

A TForm file for two documents may then look like this:

```
R^
1F^
PMr. Ron Smith^
2F^
PThe Sparkler Institute^
3F^
P16 Sparkling Road^
PSparkletown^
4F^
PUSA^
5F^
PMats G. Lindquist^
6F^
PParalog AB^
7F^
PBox 2284^
P103 17 STOCKHOLM^
P^
8F^
PSverige^
9F^
P1984-06-15^
11F
Dear Mr. Smith,
Thank you for your telex. The status of TDBS is as follows: The central modules of the
system are completed and work on the user interface is underway. We will exhibit the
system in Stockholm in November, and at that time we will have some new material about
the system, which I will send you.
The first version is, as you know, implemented on a VAX in Pascal. We will make the
system portable to other machines, e.g. IBM, in the near future.
Hoping that you can hold out a little bit longer, I remain
Yours sincerely
Mats G. Lindquist
Marketing Manager^
R^
1F^
PMats G. Lindquist^
PMats G. Löfström^
2F^
PParalog AB^
3F^
PBox 2284^
P103 17 STOCKHOLM^
4F^
PSverige^
5F^
PMr. Ron Smith^
6F^
PThe Sparkler Institute^
7F^
P16 Sparkling Road^
PSparkletown^
8F^
PUSA^
9F^
P1984-06-13
10F^
PTelex^
11F
```



TELEX NO 312/7  
ATTENTION MATS G. LINDQUIST, MATS LÖFSTRÖM  
PLEASE SEND INFORMATION ABOUT THE STATUS OF TDBS. IT IS NOW AVAILABLE ON VAX11/780? WHAT  
IS THE PURCHASE PRICE? DOES THE SYSTEM EXIST ON OTHER MACHINES?  
RON SMITH, SPARKLER INSTITUTE

The demonstration database Carroll, on the other hand, is a head-part database containing main and part records, as this extract from its SStatus information shows:

Field Name	No	Type	Part
chapter	2	PHrase	N
chaptnr	1	INteger	N
person	3	PHrase	N
speaker	4	PHrase	Y
txt	5	TExt	Y
verse	6	TExt	Y
txt2	7	TExt	Y
book	8	PHrase	N

This example shows one main record (containing all of the head fields), followed by its first two record parts:

```
R^
1F^
P6^
2F^
PPig and Pepper^
3F^
PFish Footman^
PFrog Footman^
PDuchess^
PQueen^
PPig^
PCook^
PCheshire Cat^
PMad Hatter^
PMarch Hare^
8F^
PAlice's Adventures in Wonderland^
G^
5F^
For a minute or two she stood looking at the house, and wondering what to do next, when
suddenly a footman in livery came running out of the wood - (she considered him to be a
footman because he was in livery: otherwise, judging by his face only, she would have
called him a fish) - and rapped loudly at the door with his knuckles. It was opened by a
footman in livery, with a round face and large eyes like a frog; and both footmen, Alice
noticed, had powdered hair that curled all over their heads. She felt very curious, and
crept a little way out of the wood to listen.
G^
4F^
PFish Footman^
PFrog Footman^
5F^
The Fish-Footman began by producing from under his arm a great letter, nearly as large as
himself, and this he handed over to the other, saying, in a solemn tone, "For the
Duchess. An invitation from the Queen to play croquet."
Then they both bowed low, and their curls got, entangled together
```

Alice laughed so much that she had to run back into the wood for fear of their hearing her; and, when she next peeped out, the Fish-Footman was gone, and the other was sitting on the ground near the door, staring stupidly up into the sky.^

## Updating Records with TForm

If a record in a TForm file is headed by the number of an existing BAF record, and contains nothing but fields that do not exist in the old BAF record, the new fields will be added to the BAF record.

You can also add new subfields to an already existing field. If field number two is a PHrase field, the construct:

```
2FJack^Fand^2FJill^
```

will cause three new subfields containing 'Jack', 'and' and 'Jill' to be appended to it. If the field is a TExt field, you may add new paragraphs after the last paragraph in the same way.

Should you wish to replace an old BAF record with a new TForm file record, this must be marked in the beginning of the TForm file record. For example, if you want to replace record number fifteen of your BAF file with a record beginning with the string 'Here we are.' in field number one, your record in the TForm file should look like this:

```
^15R^OR  
^1FHere we are.
```

The zero record marker will empty the old record, which will then be filled with the new contents.

To empty a field in an old record, use a zero field marker in the same way. The string:

```
^15F^OF
```

will empty field number fifteen.

To delete a record completely, without creating an empty record as the zero marker does, use a deletion marker. The string:

```
^15D
```

will delete record number fifteen. The deletion marker could either be followed by control skip characters or a record marker.

You may also use record names to identify records that are to be changed, e.g.:

```
^RJames Grieve^
```

positioned at the start of the record will cause the record with the name 'James Grieve' to be located and updated. If no such record exists, this order will be ignored.

Use the record name marker N to add a new record or update an old one. The instruction:

```
^NJames Grieve^
```

placed at the start of the record will cause a record by that name to be added, if it does not already exist.

If you are making small changes in several records at once, global updating will likely be the simplest way to change the BAF records.

## Data Type SString and the Length Marker

In a field of type SString, any characters in combination with <Ctrl> or <Esc> can be entered, and each subfield must be given with length markers specifying the length of the subfield. Each subfield part must be preceded by ^nL, with the integer n specifying the length of the part. A string subfield in several parts will be concatenated into a single subfield by the load process. A string subfield with two subfield parts containing fifteen and ten SString characters respectively could look like this:

```
^P^15Lcharscharschars^  
10Lcharschars^
```

resulting in a SString subfield containing twenty-five characters. The contents of the subfield follow immediately after the control string. The length marker is mandatory for fields of type SString, and can be used for other data types as well.

## Copying Records Using Print TForm

Records from one database can be copied to another database, using a predefined system report that creates a file in the format TForm. That file can then be loaded into other databases after any necessary editing has been done. The order is:

```
Print TForm=file.ext
```

and just as with any other Print order, Print TForm can contain a reference to a search result or to record numbers in the source database. If no extension to the file name is given, TRIP adds the extension .TFO.

If a database has received name/number/field, it is possible to specify whether the record name or number should be used in the Print TForm order. By giving the CCL order:

```
Print R TForm=file.tfo
```

the file created will then count on the string 'Nrecordname^0R' and/or '^Rrecordnumber^0R'.

## Application Software Exits (ASEs)

Application Software Exits, or ASEs, make it possible for programmers to design parts of a TRIP application in an external programming language, such as C or Fortran. ASEs are useful when:

- TRIP does not provide a function you need for your application,
- TRIP's default functionality is not powerful enough for your purposes; for example, you might need complex cross-field or cross-database validation during data entry,
- or you need to process data before it is committed to the database or to the index. This could include providing unit normalization (metric to imperial, centigrade to Fahrenheit), or lexical functions such as stem indexing, to make the searching of complex languages such as Finnish or German more intuitive.

To make this possible, TRIP defines a number of exit points which designers can use to call their own routines. Within these routines, the programmer can place calls back into the TRIP executable to gain information about the current context of the call; for instance, the database record TRIP is currently processing.

The exit points defined by TRIP follow.

### Summary

#### CCL

CAL `asename[arguments]` provides a simple exit point to a user-written routine from the CCL command line. Normally, CAL summons external products with arguments such as filename, since little contextual information is available to the routine when called in this way.

#### Output Format

`<Call(asename, item, delay)>` passes a field item (such as a subfield or a literal string) to a routine for reformatting prior to output within a text insert function. It also allows the routine to completely reformat the content of the record in memory. This is typically used to read the content of external files or fields from other databases into the current record prior to output.

#### TForm Load

This is specified during database design. On a field basis, it is used to read and possibly modify the content of an individual field or subfield. On a record basis (both before and after the record is committed), it is used to gain access to the entire record in memory, for instance, for cross-field validation.

Using TRIPmanager, the forms used for specifying the routine names are on the Advanced tab of the General Database Properties form (for record-based access), and the Advanced tab of the Field Properties form (for field-based access).

Using the TRIPapi, the routine names are specified using the base specification record fields `baffit_ase1` and `baffit_ase2` (for record-based access) and the field specification record field `baffit_ase` (for field based access).

In both cases, the values specified either on the forms or in the fields of the specification records are the names of the ASE routines.

### Index

This is specified during database design, and is used to modify the indexed values for a specific subfield or term. For instance, you may wish to index 'US' for every occurrence of the phrase 'United States', thus allowing your users to search for either variant and still find the record.

For complex languages, such as Finnish or German, morphological analysis routines can be written to index stems of terms in addition to the terms themselves, thus making searching much easier and faster. For example, in German the stem 'geschl' occurs in many terms, making the CCL search:

```
Find geschl$
```

very slow in a large database. If the stem itself were indexed, the user could simply perform the search:

```
Find geschl
```

Removing the '\$' wildcard improves search performance drastically.

Using TRIPmanager, the Advanced tab of the Field Properties form (for field-based access) can be used for specifying the routine to be called for each field.

Using the TRIPapi, specify the routine name using the field scanit\_ase in the field specification record.

In both cases, the value specified either on the forms or in the field of the specification record is the name of the ASE routine.

### Data Entry (TRIPclassic only)

This is specified during form design (TRIPclassic only), and is used on two levels to control the entry of data to a database. There are four ASEs concerned with each record, and two concerned with each field:

Record level: this is defined by pressing <kp 1> anywhere on the actual form:

- before the record is presented to the user
- after the user presses <Leave>
- after the user presses <Enter>, and before the record is committed to the database
- after the record has been committed to the database, and before the next record is presented to the user

Field level: this is defined by pressing <kp 1> while the Field Properties overlay is shown for a particular field, i.e. <Gold><kp 9> has been pressed while the cursor is in the field area:

- before entry to the field or subfield
- before exit from the field or subfield

These ASEs tend to be used for functions such as:

- Record level:
  - complex, multi-field validation
  - immediate index submission
- Field level:
  - simple cross-field validation
  - protected field manipulation
  - help messaging
  - simple data manipulation, such as conversion to and from uppercase
  - simple calculations, such as standard deviation, mean, etc.

*Note:*

*ASE invocation-sequencing conflicts may occur in the event that ASE-1 is called when entering an entry form, then ASE-2 is called when entering a field and the field associated with ASE-2 is also the first field accessed in the entry form. To overcome this, it is necessary to implement a procedure to check if ASE-1 has been executed before calling ASE-2.*

### Search Form (TRIPclassic only)

This is defined during form design, and is used for manipulation of terms in a search box prior to searching for them, for instance, converting metric units to imperial.

These ASEs are defined on page three of the search form design form by specifying a routine name in the 'ASE' column within the box specification tuple.

## The Format of an ASE Routine

All ASE routines are integer-returning functions, which take two arguments:

Argstr

Type	Character string
Access	Modify
Mechanism	By reference

Argstr is a character string, which is passed in a context-dependent manner from TRIP to the ASE. In certain circumstances the ASE can pass a value back to TRIP in the Argstr. The maximum length of the buffer which Argstr references is 256 bytes. Attempts to write more than 256 bytes to Argstr will produce unpredictable results—most likely an unrecoverable error.

Arglen

Type	Signed longword
Access	Modify
Mechanism	By reference

Arglen is a longword, which specifies the length of the character string Argstr.

The return code from the ASE to TRIP is a longword bitmask. For all ASE routines, the lowest bit (bit 0) specifies the success or failure status of the ASE routine. This bit can be set by using the manifest constants

ASE\_SUCCESS and ASE\_FAIL from the TRIPase include file (see the language-dependent sections for the actual filename). Any other bits in the return code should be set by adding the generic success-or-fail codes to the function-specific return values, such as ASE\_FIXFIELD and ASE\_REFRESH, etc. The function-specific return values are listed in the function sections that follow.

### A Template ASE in C

For C/C++ programmers, the header file to include is called TRIPASE.H, which is located in the INCLUDE directory of the TRIP tree structure.

```
#include "tripase.h"

int any_ase_name(argstr, arglen)
char *argstr;
int *arglen;
{
    ...
    return(ASE_SUCCESS);
}
```

### Linking ASE Routines to TRIP

You must build an ASE library and define a logical name to point to that library for TRIP to be able to find your ASE routines.

#### UNIX

The logical name which needs setting is called TDBS\_ASELIBS. This variable's value should contain a list of logical names mapped to the ASE libraries made using the procedure shown below, for instance:

```
MYASE1=\usr\lib\myase1
MYASE2=\usr\lib\myase2 etc.,
TDBS_ASELIBS=MYASE1,MYASE2,MYASE3 etc.
```

This variable can be set either in the user's own environment or in the system-wide tdb.conf configuration file.

To make the ASE library, use the following procedure:

- Create a directory to hold your source files.
- Copy all of the files from the ASE directory in the TRIP tree to your new directory.
- Create your source files.
- Edit the makefile (which has been copied from the TRIP ASE directory), so that the variable ASEOBJ is defined to be a list of space-separated names of the ASE object files and routine names.
- Type 'make' at the command prompt.

For example, suppose you have a TRIP installation in /usr/local/TRIP:

```
/users/dev> mkdir ase
/users/dev> cd ase
```



```
/users/dev/ase> cp /usr/local/TRIP/v31/ase/* .
```

Now suppose that you have source files 'Source1.c' and 'Source2.c' containing ASE routines 'ase1' and 'ase2':

```
/users/dev/ase> vi Makefile
... ASEOBJ=source1.o source2.o
/users/dev/ase> make
```

This will compile your source, build a TRIP jump table if necessary, and then build an executable called (by default) 'asemain'. If you have correctly defined TDBS\_ASELIBS to point to the newly-created 'asemain', you will be able to invoke ASE routines immediately.

Notes:

- *Historically, the logical name TDBS\_USRSHR was used to point to the ASE being used but, as it is only possible to specify one library with TDBS\_USRSHR, it has been deprecated, and is only retained for backward compatibility.*
- *When specifying ASE routine names, they must be lowercase only. If there are any uppercase letters in the routine name, the invocation of the routine will fail.*

## Windows

All ASE's must be compiled and linked into a DLL. The DLL must be 32-bit if you use a 32-bit TRIPsystem, and 64-bit if your TRIPsystem is 64-bit.

We recommend using a Visual Studio project file to specify the compiler and linker options for building an ASE library. An example Visual Studio 2008 solution and Visual C++ project file is available in the `ase` directory of the TRIPsystem installation.

TRIP is directed to which DLL to use by the value of TDBS\_ASELIBS in the `tdbs.conf` file. This value is a list of logical names mapped to the ASE libraries made using the procedure shown below, for instance:

```
MYASE1=c:\mylibs\myase1.dll
MYASE2= c:\mylibs\myase2.dll etc,
TDBS_ASELIBS=MYASE1,MYASE2,MYASE3 etc.
```

TDBS\_ASELIBS can be set either in the user's own copy of `tdbs.conf`, or in the system-wide `tdbs.conf` configuration file.

ASE function should be declared as below, replacing 'myase' in the example, with the name of your ASE:

```
int ASECALL myase(char*,int*)
```

Note:

*ASE names must follow the usual conventions for TRIP ASE names: max 16 characters, English alphabet letters and digits only.*

It is extremely important to remember the ASECALL macro. The windows precompiler expands it to the `__stdcall` calling convention, without which the call may suffer a fatal error. You may safely keep ASECALL in your code even if you build your ASE for other platforms as well (e.g. Linux), since its definition on non-Windows platforms is empty.



If you implement your ASE in C++, then your function must be declared as below:

```
extern "C" int ASECALL myase(char*, int*)
```

Using DEF-files to export your ASE function from the DLL is strongly recommended. The following two lines are sufficient to enable the above example function:

```
EXPORTS  
myase
```

Include the DEF file in the DLL project so that the linker will produce the DLL with the desired exports.

*Note:*

*Functions exported with decorated names (e.g. `_myase@8`), are unusable.*

## Debugging ASE routines

ASE routines can be debugged through the use of multiple `printf()` statements, or other debugging methods such as `dbx` (UNIX) and ??? (Windows).

*Notes:*

- *When debugging, it is not possible to immediately set a breakpoint in an unloaded object, however a breakpoint can be set in advance (e.g. at beginning of function) and loaded later.*
- *The ASE must be in the library being debugged.*

## CCL ASEs

The CCL statement `CAL` invokes a named ASE routine with a user specified argument string, for example:

```
CAL1 notepad This is a string to send to the ASE  
routine notepad
```

Quotation marks enclosing the argument string are not necessary. If they have been included, they will be passed unmodified to the ASE routine.

The string specified in the CCL command is passed to the ASE routine in the `Argstr` argument, with the length of the string being given by the `Arglen` argument.

*Note:*

*The argument string is not zero terminated by TRIP.*

Using the `CAL` command, the only way for the ASE routine to communicate with the calling process is via the return code from the routine. This return code can be examined using the `TRIPclassic` macro function `%RTNA` and the `TRIPapi` function `ASE RET CODE`.

The CCL interface in `TRIPclassic` supports `ASE_REFRESH`, in addition to the usual success and fail return codes. `ASE_REFRESH` can be added to either `ASE_SUCCESS` or `ASE_FAIL`, and causes the screen to be repainted upon return from your routine.

For example (the source code can be found in the SAMPLES directory in the TRIP tree, called CCLASE.C):

```
#include <stdio.h>
#include "tripase.h"
int notepad(char *argstr, int *arglen)
{
    argstr[*arglen] = '\0';
    printf("\n\n%s\n\n", argstr);
    getchar();
    return(ASE_SUCCESS + ASE_REFRESH);
}
```

## Output Format ASEs

There are two styles of ASE available within a report:

- an ASE used within a text insert function, to modify the content of an individual box, and
- an ASE used at the very top of the format specification, to allow modification of the entire record in memory. This style requires the use of the TRIPapi.

### Text Insert ASEs

Typically, text insert ASEs are used to reformat a particular value from the database, or to perform such simple functions as column addition. The ASE will be declared using a report function specification such as:

```
<box at b(*)+1,1
<t=<call(reformat, speaker.1, 0)>>
>
```

where 'reformat' is the name of the ASE routine to call, 'speaker.1' is the item from the current record to pass to the ASE routine (in this instance, the first subfield from the PHrase field speaker), and '0' is a 'delay' flag having these possible values:

- 1 no delay, call immediately
- 2 call when the user triggers a 'hot key' (normally <Gold><G> in TRIPclassic)
- 3 call after TRIP has formatted the content of a page

Instead of passing a field item to the ASE routine, a report can pass a literal string, for example:

```
<box at b(*)+1,1
<t=<call(reformat, "My String", 0)>>
>
```

To use the text insert ASE to produce effects such as column addition, you can use constructs such as:

```
<for <x>
```

```
<box at b(*)+1,1
values.x
<t=<call(add, values.x, 0)>>
>
>
<box at b(*)+1,1 <t=<call(total, "", 0)>> >
```

which will call the ASE routine 'Add' for each subfield of field values, and then call the ASE routine 'Total'.

The value specified as the item to pass to the ASE routine can be read by that routine in the Argstr argument, with the length of the item being given by the Arglen argument.

Any modifications that the ASE routine makes to the content of the Argstr string will be used by TRIP when formatting the text insert. If you do not wish anything to be output by TRIP, you must set the contents of the Arglen argument to 0 before returning from your routine.

For example:

```
#include "tripase.h"
int reformat(char *argstr, int *arglen)
{
    char *chp;
    argstr[*arglen] = '\0';
    for(chp=argstr; *chp; chp++)
        if(*chp == ' ') *chp = '_';
    return(ASE_SUCCESS);
}
```

which simply replaces all occurrences of the space character with underscores in any string passed to it.

As another example, the ASE routines 'Add' and 'Total' used in the previous format example are shown here (these routines can also be found in the SAMPLES directory in the TRIP tree, named TEXTASE.C):

```
#include <stdio.h>
#include "tripase.h"
static int current_total = 0;
int add(char *argstr, int *arglen)
{
    int iVal;
    argstr[*arglen] = '\0';
    sscanf(argstr, "%d", &iVal);
    current_total += iVal;
    *arglen = 0;
}
```

```
        return(ASE_SUCCESS);  
    }  
    int total(char *argstr, int *arglen)  
    {  
        *arglen = sprintf(argstr, "%d",  
            current_total);  
        return(ASE_SUCCESS);  
    }  
}
```

### Format-Level ASEs

If you wish to modify more than the value of a single string during output, you should use a format-level ASE. An ASE at format-level must be declared immediately after the opening chevron of a format, for example:

```
<  
    <call(format_ase, "", 0)>  
    ...  
>
```

As shown above, the call must not be placed inside a layout box.

You can only pass literal strings to this type of ASE, not field items. You can, however, gain access to the entire record in memory in the ASE routine using the function CURRENT ITEM, as documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines'. This function will return, among other things, the current record control handle pertaining to the record in memory.

Since a record control handle may only be manipulated using TRIPapi functions, you must have a TRIPapi license to modify the record in memory. If you have a TRIPapi license, you can set a cursor to the handle and retrieve or modify as you normally would.

Any modifications that you make to the record will be reflected when your ASE routine returns, with two restrictions: you will be unable to change the number of paragraphs in a TText field and the number of part records in the record.

You can work around the first restriction, however, by defining a TText field to have a maximum of one paragraph. You can then put whatever you like into that one paragraph.

### TForm Load ASEs

When you are loading data to a database using the TRIP system utility program BAFFIT, you can interact with the data before it is committed to the database. This can be very useful when performing validation beyond the scope of that provided by TRIP, or when performing complex multistage updates in many databases based on the new or updated contents of a master.

TForm load ASEs are available at two levels; field-specific ASEs and record-specific ASEs.

### Field-Specific ASEs

When you define a field-specific ASE for a database, you are telling TRIP to call your ASE routine every time that an instance of that field is encountered in the load file.

For structured field types such as PHrase, NUmber, etc., your ASE routine will be called for each distinct subfield encountered. For unstructured field types such as TExt and STring, your ASE routine will be called just once, after the field has been loaded into memory from the file.

In either case, the ASE routine names are defined in one of two ways:

In TRIPmanager, use the entry boxes on the Advanced tab of the Field Properties form (with the required field's design loaded). There you can provide the name of an ASE routine to be called during TForm load, and an ASE routine to be called during scanning. If you do not wish to call an ASE routine during scanning, only enter a value in the TForm load field.

Alternatively, use the TRIPapi to specify the name of the ASE routine to be called with the field baffit\_ase in the TRIPsystem field specification data structure (field\_spec\_rec/FieldSpecRecord).

### Structured Field-Specific ASEs

When an item from a structured field is encountered in the load file, TRIP will call your routine with the content of the item given in the Argstr and Arglen arguments. Any changes that you wish to have committed to the database should be made to these two arguments in your routine.

For example, the following ASE routine converts all lowercase letters to uppercase in the item being loaded (this example can be found in the TRIP SAMPLES directory, called TFOFIELD.C):

```
#include <ctype.h>
#include "tripase.h"
int loadase(char *argstr, int *arglen)
{
    char *chp;
    argstr[*arglen] = '\0';
    for(chp=argstr; *chp; chp++)
        *chp = islower(*chp) ? toupper(*chp) : *chp;
    return(ASE_SUCCESS);
}
```

If you wish to inhibit the loading of a particular item, you must set the length of the argument string (Arglen) to zero before returning.

If you wish to give an error message, you should return a fail code from your routine, whether or not you inhibit loading. For example:

```
#include "tripase.h"
int errorase(char *argstr, int *arglen)
{
    char msg[80];
```

```
int len;
if(... some condition ...) {
    /* Inhibit loading of item */
    *arglen = 0;
    /* Create and register error message */
    strcpy(msg, "Error in item load.");
    len = strlen(msg);
    TdbMessage(MSG_SET_ERROR, msg, &len);
    /* Trigger output of error message */
    return(ASE_FAIL);
}
}
```

To perform differing actions for the various modes (such as add, modify and delete) in which BAFFIT can operate, use the routine BAFFIT MODE documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines'. For example:

```
#include "tripase.h"
int loadase2(char *argstr, int *arglen)
{
    int mode;
    mode = TdbBaffitMode(RECORD_LEVEL);
    switch(mode) {
        case ADD_MODE      : ...
        case MODIFY_MODE   : ...
        case DELETE_MODE   : ...
    }
    return(ASE_SUCCESS);
}
```

### Unstructured Field-Specific ASEs

Unstructured fields, such as Text and SString, do not easily divide into logical 256 byte sections, and so do not permit the type of calling which is performed for structured field types.

Because of this restriction, your ASE routine is called only once for each field instance found in the load file. Consequently, whenever the field number referenced by the load file changes, your ASE routine will be called if the old field number referenced the field to which your ASE routine was attached.

For example, suppose you have attached an ASE routine to field number five of a given record design. In this instance, the following TForm layout for a single record would trigger two calls to your ASE routine, at the points marked with '\*\*\*ASE\*\*\*':

```

R^
1F^
PThis is field 1^
2F^
PThis is field 2^
5F^
PThis is the first paragraph of field 5^
PThis is the second paragraph of field 5^
3F^  ***ASE***
PThis is field 3^
5F^
PThis is the third paragraph of field 5^
4^  ***ASE***
PThis is field 4^

```

Your ASE routine would be called on the change from field five to field three, and likewise, on the change from field five to field four. Your ASE routine is not simply called once at the end of the record, and you cannot therefore assume that the entire field has been loaded once you are called (unless you know the format of the load file's content in advance).

To query the content of the field scanned or any other fields within the record, place a call to the TRIPsystem function CURRENT ITEM to gain the current record control handle (as documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines'). This handle can then be interrogated and updated using a TRIPsystem cursor. You must have purchased a TRIPapi license to do this.

To inhibit the loading of the field in question, you must explicitly delete the content of that field using the TRIPapi call DELETE ITEM.

If you wish to provide an error message, you should return a fail code from your routine whether or not you inhibit loading. For example:

```

#include "tripase.h"

int errorase(char *argstr, int *arglen)
{
    char msg[80];
    int  len;
    if(... some condition ...) {
        /* Create and register error message */
        strcpy(msg, "Error in item load.");
        len = strlen(msg);
        TdbMessage(MSG_SET_ERROR, msg, &len);
        /* Trigger output of error message */
        return(ASE_FAIL);
    }
}

```

```
}  
}
```

To perform differing actions for the various modes ( add, modify and delete) in which BAFFIT can operate, use the routine BAFFIT MODE documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines'. For example:

```
#include "tripase.h"  
int loadase2(char *argstr, int *arglen)  
{  
    int mode;  
    mode = TdbBaffitMode(RECORD_LEVEL);  
    switch(mode) {  
        case ADD_MODE      : ...  
        case MODIFY_MODE   : ...  
        case DELETE_MODE    : ...  
    }  
    return(ASE_SUCCESS);  
}
```

### Record-Specific ASEs

Record-specific TForm load ASEs normally perform complex cross-field validation exercises beyond the scope of TRIP's default operators.

Two record-level access ASEs have been defined: before and after the record is committed to the database. In both cases, the only access mechanism for the record is via the current record control handle, gained by calling the TRIPsystem function CURRENT ITEM (as documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines'). Again, you must have purchased a TRIPapi license to do this.

To define the names of the ASE routines to be called:

- 1 In TRIPmanager, enter the ASE to be called, both before and after commit, in the 'Data Loading' section on the 'Advanced' tab of the General Database Properties form. You will be prompted for the names of the routines to be called.
- 2 With the TRIPapi, use the fields baffit\_ase1 and baffit\_ase2 in the TRIPsystem database specification data structure (base\_spec\_rec/BaseSpecRecord) to specify the names of the routines to call before and after the commit, respectively.

If you wish to inhibit the loading of a record, you should return a fail code from your ASE routine.

To issue an error message, call the TRIPsystem callback function MESSAGE prior to returning a fail code. You cannot use this mechanism for delivering an error message if you return a success code. For example:

```
#include "tripase.h"  
int recordase(char *argstr, int *arglen)
```



```
{
    char msg[80];
    int len;
    if(... some condition ...) {
        strcpy(msg, "Cannot load record.");
        len = strlen(msg);
        TdbMessage(MSG_SET_ERROR, msg, &len);
        return(ASE_FAIL);
    }
    return(ASE_SUCCESS);
}
```

To perform differing actions for the various modes ( add, modify and delete) in which BAFFIT can operate, use the routine BAFFIT MODE. For example:

```
#include "tripase.h"
int loadase2(char *argstr, int *arglen)
{
    int mode;
    mode = TdbBaffitMode(RECORD_LEVEL);
    switch(mode) {
        case ADD_MODE      : ...
        case MODIFY_MODE   : ...
        case DELETE_MODE   : ...
    }
    return(ASE_SUCCESS);
}
```

## Index ASEs

You can specify which terms are to be indexed (either to exclusion of the terms within the actual data, or in addition) by interacting with TRIP when it is preparing entries for the index file.

For example, you may wish to have the term 'United States' indexed wherever the term 'US' occurs within the database. Users can then search for either, and find both.

When processing languages with a high degree of complexity, such as Finnish or German, you can determine which terms should be indexed in their entirety and which should be indexed by their stems. For example, in German the stem 'geschl' occurs in many terms, and so in a large database the search:

```
Find geschl$
```

will be relatively slow in completing. As this is a very useful type of search, an index ASE can be used to direct the index engine to add the stem 'geschl' to the index at every point where a derived term occurs, such as 'geschlossen'.

An index ASE can only be defined on a per-field basis. Your ASE routine will be called for each term which occurs in that field, with the term specified in the Argstr/Arglen parameters. Any changes that you make to these parameters will be reflected in the index files, according to the circumstances detailed below.

To define the names of the ASE routine to be called:

- In TRIPmanager, user the entry boxes on the Advanced tab of the Field Properties form to specify the names of the routines to be called during TForm load and scanning. Specify the scanning ASE if you wish your routine to be called during Index.
- With TRIPapi, use the field scanit\_ase in the TRIPsystem field specification data structure (field\_spec\_rec /FieldSpecRecord) to specify the name of the routine to call during Index.

The conventions used for the Index ASE are slightly different, depending on the type of field to which the ASE routine is attached.

If the field is of type TExt, NUmber, INteger, DAte or TIme, your routine will be called for each term which is scanned in that field. If you want your routine to have only the original term indexed, then:

- do not modify the contents of Argstr
- set Arglen to zero before return
- return ASE\_SUCCESS from your routine

If your routine should have new terms indexed instead of the original, then:

- modify the contents of Argstr to the new term(s) required
- set Arglen to the length of the new term(s)
- return ASE\_FAIL from your routine

If you want your routine to have new terms indexed as well as the original, then:

- modify the contents of Argstr to the new term(s) required
- set Arglen to the length of the new term(s)
- return ASE\_SUCCESS from your routine

If you are specifying more than one term, either in addition to the original or as a replacement, the terms should be separated by one space character.

If the field being scanned is of type PHrase, your routine will also be called with the entire subfield as well as with each component term. When TRIP has

written an entire subfield into Argstr, Arglen will be negative, to signal the difference between the two.

If Arglen is negative, your routine can have just the original phrase indexed by:

- setting Arglen to zero before return
- not modifying the contents of Argstr
- returning ASE\_SUCCESS

If Arglen is negative, you can have a new phrase indexed instead of the original by:

- modifying the contents of Argstr
- setting Arglen to the length of the new phrase
- returning ASE\_FAIL

If Arglen is negative, you can also have both a new phrase and the original indexed by:

- modifying the contents of Argstr
- setting Arglen to the length of the new phrase
- returning ASE\_SUCCESS

If Arglen is positive, you can have just the original term indexed by:

- not modifying the contents of Argstr
- setting Arglen to zero before return
- returning ASE\_SUCCESS from your routine

If Arglen is positive, you can have new terms indexed instead of the original by:

- modifying the contents of Argstr to the new term(s) required
- setting Arglen to the length of the new term(s)
- returning ASE\_FAIL from your routine

If Arglen is positive, you can also have new terms indexed as well as the original by:

- modifying the contents of Argstr to the new term(s) required
- setting Arglen to the length of the new term(s)
- return ASE\_SUCCESS from your routine

Thus, your routine could be called for any of the following:

- the original phrase subfield
- each term within the original subfield
- each term within a replacement for the original subfield

For example (this example can be found in the SAMPLES directory of the TRIP tree, called SCANASE.C):

```
#include "tripase.h"
int indexase(char *argstr, int *arglen)
{
    if(*arglen < 0) { /* entire subfield */
        argstr[-(*arglen)] = '\0';
        if(!strcmp(argstr, "UNITED STATES")) {
            /* Accept United States without modification */
            *arglen = 0;
            return(ASE_SUCCESS);
        }
        else if(!strcmp(argstr, "GREAT BRITAIN")) {
            /* Add "United Kingdom" to "Great Britain" */
            strcpy(argstr, "United Kingdom");
            *arglen = strlen(argstr);
            return(ASE_SUCCESS);
        }
        else if(!strcmp(argstr, "TIMBUKTU")) {
            /* Replace Timbuktu with "Where?" */
            strcpy(argstr, "Where?");
            *arglen = strlen(argstr);
            return(ASE_FAIL);
        }
    }
    else { /* single term */
        argstr[*arglen] = '\0';
        if(!strcmp(argstr, "UNITED")) {
            /* Replace "united" with "divided" */
            strcpy(argstr, "divided");
            *arglen = strlen(argstr);
            return(ASE_FAIL);
        }
    }
    /* Catch all - no new terms, index original */
    *arglen = 0;
    return(ASE_SUCCESS);
}
```

This example will:

- 1 Allow 'United States' to be indexed as an entire phrase
- 2 Add 'United Kingdom' wherever 'Great Britain' occurs
- 3 Replace 'Timbuktu' with the WHERE?
- 4 Replace the term 'United' with the term 'Divided'

This will have several effects:

- Field-specific searches for 'United States' will fail, unless the search term is single-quoted:

```
Find MYPHRASE = UNITED STATES      - No hits!
Find MYPHRASE = 'UNITED STATES'    - Hits
```

This is because the phrase 'United States' was indexed, but the individual term 'United' was replaced with 'Divided'. Thus, the following search will find records containing 'United States':

```
Find MYPHRASE = DIVIDED STATES - Hits
```

- Searching for 'United Kingdom', with or without single quotes, will locate records containing 'Great Britain'.
- Searching for 'Timbuktu' will always fail, but searching for WHERE? will hit records containing 'Timbuktu'.

### Data Entry ASEs (TRIPclassic only)

There are six types of ASE defined for data entry forms, none of which pass any arguments to the ASE routines. All interaction with the data onscreen, or in the record in memory, must be performed using a set of specialized routines for TRIPclassic interaction or by using the TRIPapi.

The six ASEs defined are:

- 1 On initialization of the form prior to the user being allowed to input.
- 2 On the user leaving the form via a <Leave> action, e.g. <PF 3>.
- 3 On the user committing the record using <Enter>, before the record is actually written to the database.
- 4 After the record has actually been written to the database and before the initialization ASE is invoked once more (if further data entry is to be performed).
- 5 On entry to a particular field box.
- 6 On exit from a particular field box.

To define the routines to be called at a form-based point (numbers one through four above), press <kp 1> at any time when the field properties overlay is not shown during data entry form design. You will be prompted to supply up to four ASE routine names.

To define the routines to be called at a box-based point (numbers five and six above), press <kp 1> when the field properties overlay is shown, i.e. you have pressed <Gold><kp 9> in an attached field box. You will be prompted to supply up to two ASE routine names.

There are a number of return code bit settings that are specific for data entry ASE routines:

Setting Name	Function
ASE_REFRESH	signals TRIP to repaint the screen on return from your routine
ASE_CONTINUE	signals TRIP to simulate a repeat of the keystroke which occurred just before the invocation of your routine
ASE_MESSAGE	signals TRIP to report a message registered using the MESSAGE callback function (useful when you want to report a message without returning a fail code)
ASE_FIXFIELD	signals TRIP to leave the cursor in the box to which you have set it, rather than simply moving to the next in sequence
ASE_NOFIELD	signals TRIP to disallow any user input to the form

There are also a number of TRIPclassic specific callback functions, summarized below and documented fully in the section entitled 'TRIPclassic Callback Functions for ASE Routines':

Function Name	Purpose
CHECK ENTRY	returns to your routine the field number and item, or row, number at which the cursor is currently positioned
GET LINE	returns the content of the line in which the cursor is currently positioned
PUT LINE	overwrites the content of the line in which the cursor is currently positioned
SET ENTRY	sets the cursor to a specific field and item, or row, number
WRITE MESSAGE	delivers a message on the TRIP message line immediately, rather than on return from your routine as is the case with the MESSAGE callback

### Form-Based ASEs (TRIPclassic only)

The form-based ASEs (points one through four of the Data Entry ASE list given previously) do not allow onscreen modification or interrogation of data. If you wish to change or read the contents of the record in a routine invoked from one of these ASEs, you must use the TRIPapi to do so. In this case, you can use the TRIPsystem function CURRENT ITEM to get the current record control handle, which can be manipulated using a standard TRIPsystem cursor.

The only interaction that can be performed with TRIPclassic is via the return code from your ASE routine, as detailed below.

### Form Initialization (TRIPclassic only)

This ASE is called before the user is actually allowed to see the data entry form. Its normal use is therefore either to initialize data for the later ASEs or to stop the user from having access to the form.

When initializing data, your routine should always return a success code. When preventing form access, your routine should always return a fail code. TRIP will only act upon this code, however, if the user has attempted to enter data entry with the CCL EEdit command. If the user has entered data entry via the standard menus, the return code will have no effect.

Typically, if you are preventing the form from appearing, your routine will call the MESSAGE function from the TRIPsystem to report to the user why his or her EEdit command has failed. This is documented in the section entitled 'TRIPsystem Callback Functions for ASE Routines',

### Quitting the Form Using <Leave> (TRIPclassic only)

TRIP invokes this ASE when the user makes modifications to the record onscreen and presses either <Leave> or <Gold><Leave>. Returning a fail code from your routine at this point will stop the quit action from completing, i.e. it will keep the user in the form.

Treat this ASE with care. Making it impossible for the user to quit data entry will result in many records of poor quality being committed to the database, since <Enter> will then be the only permissible method for leaving data entry. To avoid the confusion generated by non-working keystrokes, be sure to provide appropriate messages when such circumstances arise.

### Record Commit Before Writing to BAF

TRIP invokes this ASE when the user submits the record, signalling that all modifications have been completed. You can interrupt the sequence, however, by returning a fail code from your ASE routine, as the record has not yet been written to the database.

If you do return a fail code, you will probably want to direct the user to a particular field for update. You can do this with the callback function SET ENTRY, as documented in the section entitled 'TRIPclassic Callback Functions for ASE Routines', and setting the FIXFIELD bit in the ASE routine return code. For example:

```
#include "tripase.h"
int prewritease(char *argstr, int *arglen)
{
    char msg[80];
    int  len;
    if(... some condition ...) {
        /* Format and register error message */
        strcpy(msg, "Bad value in field");
        len = strlen(msg);
        TdbMessage(MSG_SET_ERROR, msg, &len);
        /* Move the cursor to the incorrect field
        (26) */
        TedSetEntry(26, 1);
        /* Signal TRIP to leave cursor in place
        */
    }
}
```

```
        return(ASE_FAIL + ASE_FIXFIELD);
    }
    return(ASE_SUCCESS);
}
```

If you do not use the FIXFIELD bit in the return code, TRIP will place the cursor in the first box on the entry form and ignore any field placement performed by the SET ENTRY function.

### Record Commit After Writing to BAF

TRIP invokes this ASE once the record has been successfully written to the database. The ASE will not be invoked if the commit failed.

If your routine returns a success value, the user can continue to the next record or return to CCL.

If your routine returns a fail value, the record has already been written but the data entry mode is set to 'modify', giving the user the ability to edit it. Additional record commits of this same record will modify the record further, rather than adding a new record to the database.

For example:

```
#include "tripase.h"
int postwritease(char *argstr, int *arglen)
{
    if(... some condition ...) {
        /* Create and register a message */
        strcpy(msg, "You must update this value");
        len = strlen(msg);
        TdbMessage(MSG_SET_ERROR, msg, &len);
        /* Set the cursor to the required field */
        TedSetEntry(26, 1);
        /* Return fail - switch TRIP to modify mode */
        return(ASE_FAIL + ASE_FIXFIELD);
    }
    return(ASE_SUCCESS);
}
```

If you have a TRIPapi license, you can delete the record just created with the TRIPsystem functions CURRENT ITEM and DELETE RECORD.

### Box-Based ASEs (TRIPclassic only)

The box-based ASEs (points five and six of the Data Entry ASE list given previously) allow onscreen modification of data. By using the routines documented in the section entitled 'TRIPclassic Callback Functions for ASE Routines', any changes in your ASE routine will be discernible to the user at the time of modification.



TRIP invokes box-level ASE routines differently for TExt fields than for other field types. If the field in question is of type PHrase, NUmber, INteger, DAtE or TIme, each of the box-based ASEs will be invoked separately for each subfield in which a modification is made.

During data add, the entry ASE will be invoked when the cursor is first placed in the box, and the exit ASE will be invoked when the cursor either leaves the box or is moved to the next subfield using <Return>. If a new subfield is to be added, the entry ASE is called again before the user can enter the subfield.

For a TExt field, the entry ASE is invoked once on entry to the box, and the exit ASE is invoked once on exit from the box if the user has made any modifications to the content of that box.

To enable your routine to make onscreen data modifications, you should place calls to the TRIPclassic callback functions GET LINE and PUT LINE. These act on the 'current' field and row set using the function SET ENTRY.

To protect a particular box from a certain class of user but not from all users, set the ASE\_CONTINUE bit in the return code. This bit causes TRIP to simulate a repeated keystroke, for example, as if the user had pressed the <Tab> key twice to skip over a field.

### Search Form ASEs (TRIPclassic only)

The only ASE defined for search forms is used for each search box on the form. This ASE is defined on page three of the layout screen in the ASE column of the box specification tuple.

When defined, this ASE will be invoked when the user leaves the box in question. Your ASE routine can then use the TRIPclassic callback function GET LINE to retrieve the data input by the user. Your routine can use PUT LINE to replace that data following data modifications, and can also modify the content of any other search box on the screen by using SET ENTRY before PUT LINE.

When calling SET ENTRY, the 'field number' should be the ordinal box number as defined on page two of the layout screen, and the 'row number' should always be set to 1.

You cannot use the FIXFIELD bit in the return code on a search form, as TRIP will ignore any attempt to set the real cursor to another box out of <Tab> sequence.

### TRIPsystem Callback Functions for ASE Routines

Within an ASE routine, it is often useful to be able to place a call into TRIPsystem to establish the user's current context, or to report a message in a standard manner.

### TRIPclassic Callback Functions for ASE Routines

Within an ASE routine, it is often useful to be able to place a call into TRIPclassic to perform such functions as writing data to field boxes in data entry, or issuing messages before your routine returns.

If you have purchased a TRIPapi license, you can use all of the TRIPapi functions from within ASE routines. If you have not, the following pages detail those routines which are available to all ASE programmers.

## TRIP API Reference Guide

Refer to the Toolkit Reference Manual for details on all TRIPsystem API calls. It is supplied with the TRIPsystem distribution as a ZIP-compressed set of HTML based documentation.



## List of Figures and Tables

### Figures

Figure 1–2	The CONTROL database	20
Figure 1–3	Head and part records in a database	21
Figure 1–4	Carroll's head/part record structure	22
Figure 1–5	A head record	22
Figure 1–6	A part record	22
Figure 1–7	A record entity	22
Figure 1–8	A composite record	23
Figure 1–9	Record components	23
Figure 2–1	New Database Wizard	25
Figure 2–2	New Database General Properties	26
Figure 2–3	Database Name Entry Field	26
Figure 2–4	The Database File Location Selection Boxes	27
Figure 2–5	Transaction log selection	28
Figure 2–6	XML Enabling a Database	29
Figure 2–7	The Database Description field	29
Figure 2–8	New Database Design Wizard Completion page	30
Figure 2–9	DB Creation Confirmation	30
Figure 2–10	Specify Field Collection Query	30
Figure 2–11	The Database General Properties Form	31
Figure 2–12	Sample SYSTEM default report, 'Dump'	32
Figure 2–13	The Database Files Properties Form 1	34
Figure 2–14	The Database Files Properties Form 2	35
Figure 2–15	The Database Indexing Properties Form	36
Figure 2–16	Natural Language Treatment selection box	36
Figure 3–1	The 'Train' thesaurus, vertical representation	81
Figure 3–2	The 'Train' thesaurus, horizontal representation	81
Figure 3–3	New Thesaurus Menu	83
Figure 3–4	STatus for thesaurus 'Thesali'	88
Figure 5–1	Entry forms for database CORR	101
Figure 5–2	Properties for CORR entry form FULL	102
Figure 5–3	Copy a Data Entry form	102
Figure 5–4	Name New Data Entry Copy	103
Figure 5–5	Data Entry Copy Confirmation	103
Figure 5–6	Delete a Data Entry form	103
Figure 5–7	Delete Data Entry form confirmation	104
Figure 5–8	Data Entry form Deleted	104
Figure 6–1	Report layout and construction	106
Figure 6–2	Report components	106
Figure 6–3	New Report Menu	108
Figure 6–4	New Output Format name entry dialog	108
Figure 6–5	New report Properties dialog	109
Figure 6–6	New report Content dialog	110
Figure 6–7	Paged output	140
Figure 6–8	The Show Format window	143
Figure 7–1	Search forms for a TRIP installation	203
Figure 7–2	Properties for search form ALICE_DEMO	203
Figure 7–3	Copy a Search Form	204
Figure 7–4	Name Search Form Copy	204

Figure 7–5	Search Form copy confirmation	204
Figure 7–6	Delete a Search form	205
Figure 7–7	Delete Search Form confirmation	205
Figure 7–8	Search form Deleted	205
Figure 9–1	Indexing the Database TestThes	218
Figure 9–2	Load a TForm file into database TestThes	219
Figure 9–3	Load TForm Specify File Name form	219
Figure 10–1	Creating a New User	225
Figure 10–2	The create New User form	225
Figure 10–3	The User created confirmation dialog	226
Figure 10–4	Deleting the user 'Fred'	226
Figure 10–5	The Delete User Confirmation	227
Figure 10–6	The Deleted User Access Loss Confirmation	227
Figure 10–7	Opening Properties for the User, FREDERICO	227
Figure 10–8	The user FREDERICO's user Properties form	228
Figure 10–9	Date Format selection box	228
Figure 10–10	Ignore TRIP password checkbox	228
Figure 10–11	Date Format selection box	229
Figure 10–12	Date Format Selections	229
Figure 10–13	Changing the date digit separator	229
Figure 10–14	Management privilege settings	229
Figure 10–15	Session parameter settings	230
Figure 10–16	Company information entry area	230
Figure 10–17	Procedures for user FREDERICO	231
Figure 10–18	Group membership for user FREDERICO	231
Figure 10–19	The Add To Group form	232
Figure 10–20	Access Rights for user FREDERICO	232
Figure 10–21	Creating a New Group	233
Figure 10–22	New Group dialogue	233
Figure 10–23	New Group Created Confirmation	233
Figure 10–24	Deleting a group	234
Figure 10–25	Confirming deletion of a group	234
Figure 10–26	The 'My users' sub-tree	235
Figure 10–27	The Add Group Member confirmation	235
Figure 10–28	The Delete Member confirmation	235
Figure 10–29	The Change Manager option	236
Figure 10–30	Change Manager Selection box	236
Figure 10–31	Change Manager Confirmation	236
Figure 11–1	Granting Access to Database CARROLL	238
Figure 11–2	The Access Level Form	239
Figure 11–3	Database Name Selection	239
Figure 11–4	Database Name Selection	239
Figure 11–5	Field and Record Restrictions	240
Figure 11–6	Record-level READ rights for 'FREDERICO'	242
Figure 11–7	Record-level WRITE rights for 'FREDERICO'	242
Figure 11–8	The Change Manager action menu option	245
Figure 11–9	Change Manager Selection	245
Figure 11–10	Carroll's Show ACcess screen	246

## Tables

Table 0–1	TRIP naming conventions	12
-----------	-------------------------	----

Table 1–2	Sample flat file table	15
Table 1–3	Sample relational database tables	16
Table 1–4	Sample full-text database table	17
Table 2–1	Special characters	38
Table 2–2	The character folding classes	39
Table 2–3	Truncation, masking and special symbols	40
Table 2–4	The character classes	42
Table 2–5	Paragraph definition in TRIP	46
Table 2–6	Sentence definition in TRIP	48
Table 2–7	Field Defaults and Restrictions	55
Table 2–8	Use of the record name field	60
Table 2–9	Symbols used in pattern specification	64
Table 2–10	TRIP's predefined character sets	65
Table 2–11	A simple pattern	65
Table 2–12	A more complex pattern	66
Table 2–13	More patterns	67
Table 2–14	Sample combined character sets	68
Table 2–15	Modifying a database design	71
Table 3–1	Record contents and thesaurus design for 'Train'	82
Table 3–2	The thesaurus template	84
Table 3–3	Record contents and thesaurus design	85
Table 3–4	Hierarchical relationships of the 'Train' thesaurus	86
Table 4–1	Hierarchical relationships of the 'Train' thesaurus	93
Table 4–2	A sample accounting file	95
Table 6–1	Types of background text	121
Table 6–2	Text string reserved characters	121
Table 6–3	Headers	122
Table 6–4	Separators	122
Table 6–5	Trailers	125
Table 6–6	Text string functions	127
Table 6–7	Field type-dependent functions	129
Table 6–8	Box and box group functions	133
Table 6–9	Format functions	134
Table 6–10	FOR loop functions	135
Table 6–11	Structure of Olympic_Games	137
Table 6–12	Date formats	158
Table 6–13	Samples of <Numform> output	186
Table 8–1	Anatomy of a global update command	208
Table 8–2	Structure of a global update using record numbers	208
Table 8–3	Generic update targets	209
Table 8–4	Structure of a global update using a search result	212
Table 8–5	Record update targets	212
Table 9–1	Operating systems and log file names	220
Table 9–2	Running the BAFINI utility	221
Table 11–1	General field access rights	240
Table 11–2	Unsupported combinations of access rights	241

## Index

- symbol .....	41, 64, 226, 238
' symbol .....	40, 44
! symbol .....	40, 44, 121, 276
" symbol .....	40
# symbol .....	40
\$ symbol .....	40, 282
%RTNA .....	286
& symbol .....	40
( symbol .....	44
() symbol .....	40, 64
) symbol .....	48
* symbol .....	64
. symbol .....	40, 44, 226, 276
.. symbol .....	64
/	
symbol .....	40
/ symbol .....	64, 121, 226
// symbol .....	64
/symbol .....	44
: symbol .....	47, 226
? symbol .....	40, 44, 276
[ symbol .....	44
] symbol .....	48
^ symbol .....	273
_ symbol .....	26, 121
{ symbol .....	44
} symbol .....	48
+ symbol .....	40, 64
< symbol .....	44, 105, 110, 121
<_>, as convention .....	11
<Append> .....	135, 144
<At_end> .....	133, 146
<Base> .....	127, 147
<Call> .....	281
format .....	134, 148
text string .....	127, 150
<Case> .....	133, 151
<Chr> .....	127, 153, 154
<CR> .....	42
<CR>, as convention .....	11
<CR><LF> .....	275, 276
and TForm .....	274
<Curdate> .....	127, 155
<Dateform> .....	127, 156
<Debit> .....	70, 134, 158
<Ff> .....	127, 159
<FF> .....	42
<FF>, as convention .....	11
<Fieldname> .....	129
<Fieldno> .....	129
<Fieldtype> .....	129
<FOR> loops .....	161
<Gold>	
<Gold><G> .....	287
<Gold><kp 9> .....	282, 298
<Gold><Leave> .....	300
<Hitlist> .....	135, 164
<Hits> .....	127, 166
<If-changed> .....	133, 167
<If-empty> .....	133, 169
<If-nonempty> .....	133, 170
<If-unchanged> .....	133, 171
<Indent> .....	107, 133, 173
<kp 1> .....	282, 298
<Leave> .....	300
<LF> .....	42, 45, 48, 61, 276
<LF>, as convention .....	11
<Link> .....	133, 175
<Loop variables> .....	135
<NL>, as convention .....	11
<Noff> .....	134, 178
<Nolf> .....	134, 179
<Noorig> .....	107, 133, 180
<Numform> .....	127, 182
<Occs> .....	127, 184
<Once> .....	133, 185
<Orig> .....	133, 186
<Pageno> .....	127, 188
<Paragraphno> .....	129
<Parts> .....	127, 189
<PF3> .....	298
<Rid> .....	127, 190
<Ris> .....	127, 191
<Rname> .....	127, 192
<Sentenceno> .....	129
<Sortfields> .....	106, 134, 193
<SP>	
and TForm .....	274
<Subfieldno> .....	129
<Subrid> .....	127, 194
<Substring> .....	127, 195
<Tab> .....	61, 276, 302
<Text variables> .....	134, 196
<Timeform> .....	127, 198
<Trace> .....	133, 197
<VT> .....	42
<Weight> .....	127, 199
> symbol .....	48, 105, 110, 121
Access	
database	
cluster .....	240
defining .....	236



field level .....	238	FAIL .....	286
first form .....	238	field-specific .....	290
general field .....	237	structured .....	290
hidden read scope .....	240	unstructured .....	291
hierarchy of access rights .....	240	form initialization (TRIPclassic only) .....	299
listing .....	242	format .....	283
read .....	237	format-level .....	289
read scope .....	237, 238, 240	form-based .....	298
record level .....	238	form-based (TRIPclassic only) .....	299
write .....	237	index .....	282, 294
write scope .....	237, 238, 239, 240	library .....	284
print .....	243	linking to TRIP .....	284
show .....	242	quitting form with <Leave> (TRIPclassic only) .....	300
Access privileges		record commit	
database .....	235	after writing to BAF .....	300, 301
Accounting log		record-specific .....	293
B-line .....	93	reports .....	281, 287
C-line .....	93	RET CODE .....	286
E-line .....	93	scanit_ase .....	282
F-line .....	93	search form (TRIPclassic only) .....	283, 302
M-line .....	93	template, in C .....	284
O-line .....	93	text insert .....	287
Q-line .....	93	TForm load .....	281, 289
R-line .....	93	TRIPclassic callback functions .....	302
S-line .....	93	TRIPkernel callback functions .....	302
U-line .....	93	uses of .....	281
Added fields		ASE_	
thesaurus .....	87	CONTINUE .....	299, 302
Adding		FAIL .....	284, 295, 296
user group member .....	232	FIXFIELD .....	284, 299
Administrator		MESSAGE .....	299
database .....	220	NOFIELD .....	299
system .....	220	REFRESH .....	284, 286, 299
Alice database .....	10, 11, 32	SUCCESS .....	284, 286, 295, 296
Status .....	73	ASEOBJ .....	284
Arglen .....	283, 286, 288, 290, 295, 296	Background text	
Argstr .....	283, 286, 288, 290, 295, 296	reports .....	120
ASE		BAF .....	23, 204, 214
baffit_		and general database properties .....	27
ase .....	281	and LOAD procedure .....	273
ase1 .....	281	and TForm .....	279
ase2 .....	281	and the record name field .....	60
box-based .....	298	BAFFINI .....	218
entry .....	302	BAFFIT .....	289, 291, 293, 294
exit .....	302	MODE .....	291, 293, 294
box-based (TRIPclassic only) .....	301	Baffit_	
CCL .....	281, 286	ase .....	281, 290
data entry		ase1 .....	281, 293
field level .....	282, 283	ase2 .....	281, 293
record level .....	282, 283	Base access	
data entry (TRIPclassic only) .....	282, 298	print .....	243
debugging .....	286	show .....	242
directory .....	284		

Base file .....	23	<once> .....	133, 185
Base File .....	see also BAF	<orig> .....	133, 186
Base index file .....	23	<trace> .....	133, 197
Base Index File .....	see also BIF	CALI .....	281, 286
Base_spec_rec .....	293	Carriage return	
BaseSpecRecord .....	293	reports .....	121
Batch		Carroll database .....	10
update .....	10	chapter information in .....	21
BIF .....	23, 204	page information in .....	21
and general database properties .....	27	records	
and the record name field .....	60	chapter .....	22
Bigram .....	24	main .....	22
B-line .....	93	paragraph .....	22
Bold, as convention .....	11	part .....	22
Box .....	106	STatus .....	278
constituents .....	106	Case sensitivity	
definition .....	106	and global updating .....	213
functions .....	133	CCL	
group		commands and reports .....	143
definition enclosures .....	119	search menu option .....	11
reports .....	119	CCLASE.C .....	287
layout		Character folding class	
defining a .....	110	and diacritics .....	38
numbering .....	113	and umlauts .....	38
page level .....	140	default .....	38
positioning .....	113	ENGLISH .....	38
using coordinates .....	113	specification .....	37
using preceding boxes .....	114	SWEdish .....	38
proportioning		Character masks, as searchable characters .	40
using columns .....	118	Character sets	
using lines .....	117	specification .....	64
using lines and columns .....	116	Characters	
reports		control skip .....	274
header .....	140	ignore search characters .....	48
trailer .....	141	paragraph separators .....	45
simple .....	110	paragraph start .....	45
size		paragraph terminators .....	46
reports .....	116	reserved .....	121
specifications		searchable .....	40
directed .....	113	special .....	38
nonspecific .....	113	CHECK ENTRY .....	299
Box/box group		Chevron, as convention .....	11
functions		C-line .....	93
<append> .....	144	Cluster	
<at_end> .....	133, 146	creating .....	75
<case> .....	133, 151	deleting .....	78
<if-changed> .....	133, 167	modifying .....	76
<if-empty> .....	133, 169	Columnar output .....	142
<if-nonempty> .....	133, 170	Command	
<if-unchanged> .....	133, 171	DEfine .....	11
<indent> .....	133, 173	EForm .....	33
<link> .....	133, 175	Format .....	33
<noorig> .....	133, 180	INdex .....	215



Component		deleting.....	103
in head/part database .....	23	Database	
Composite record.....	21, 23	access	
Constituent		cluster .....	240
box .....	106	defining .....	236
CONTROL database.....	20, 221, 223	field level.....	238
contents.....	20	first form.....	238
Control master, and TForm .....	274	general field .....	237
Control skip characters .....	274	hidden read scope.....	240
Control strings		hierarchy of access rights.....	240
and TForm.....	273	listing.....	242
Conventions		privileges.....	235
<~> .....	11	read.....	237
<CR> .....	11	read scope .....	237, 238, 240
<FF> .....	11	record level .....	238
<LF> .....	11	write .....	237
<NL>.....	11	write scope.....	237, 238, 239, 240
boldface.....	11	administration .....	10
chevrons.....	11	administrator .....	20
Courier fonts.....	11	Alice .....	10, 32
italic .....	11	STatus .....	73
lower case .....	11	Carroll .....	10
naming.....	12	chapter information in.....	21
space character .....	11	paragraph information in .....	21
upper case.....	11	records	
Copying		chapter.....	22
records with TForm.....	280	main.....	22
reports .....	107	paragraph .....	22
with global updating .....	212	part .....	22
Corr database .....	10	STatus .....	278
and reports .....	110	cluster	
database field numbers.....	276	reports and.....	142
structure .....	276	CONTROL.....	20, 221, 223
Courier fonts, as convention .....	11	contents .....	20
Creating		Corr .....	10
reports .....	107	database field numbers .....	276
search forms.....	201	description	
user .....	222	and general database properties.....	29
user group .....	229	and STatus.....	29
Current		design	
date form .....	226	copying .....	72
CURRENT ITEM.....	289, 292, 293, 299, 301	deleting .....	72
Data		modifying.....	71
models.....	15	saving .....	71
normalization .....	16	field numbers.....	276
organization, and TRIP .....	17	general properties .....	26
Data entry		head/part	
database description.....	34	component .....	23
entry form .....	101	head record.....	22
default .....	33, 34	part record.....	22
Data Entry Forms.....	101	record.....	23
copying .....	102	record entity .....	22
creating.....	102	management system	

full-text.....	15, 16	Delineators, as searchable characters.....	40
reindexing.....	217	Digits, as searchable characters.....	40
relational.....	15	Directory	
responsibility, transferring.....	242	ASE.....	284
security.....	10	INCLUDE.....	284
Thesali.....	10	SAMPLES.....	287, 288, 290, 297
Thesauri.....	80	Document, and TRIP.....	18
TRIP basics.....	20	Dump reports.....	130
what is a thesaurus?.....	80	Element	
Database administrator.....	220	reports.....	106, 120
Database Administrator..see also File Manager		E-line.....	93
Database Cluster		ENGLISH	
creating.....	75	character folding.....	38
deleting.....	78	Entity	
modifying.....	76	record.....	22
Database Corr		Entry form	
structure.....	276	data entry.....	101
Databases		ERRLOG.....	217
listing.....	74	Error checking	
Date		global updating.....	214
form, current.....	226	F marker.....	273, 275
DAte.....	19	Field	
restrictions.....	69	accounting.....	69
DEBIT.LOG.....	70, 90, 91	attributes.....	60
Default		create new.....	57
data entry form.....	33, 34	database reference.....	62
reports.....	32	defaults and restrictions.....	55
Define		define ASE.....	69
space character.....	40	define pattern.....	63
DEfine.....	11	delete.....	204
EForm.....	33	edit or delete.....	56
Format.....	33	elements	
Definition		and reports.....	112
box.....	106	head.....	21
Delete.....	204	index mode.....	58
field.....	204	index settings.....	58, 59
paragraph.....	204	insert.....	204
record.....	204	layout retained.....	61
sentence.....	204	list.....	57
string.....	204	list of.....	57
subfield.....	204	make part field.....	61
DElete.....	206	mandatory.....	61
DELETE ITEM.....	292	marker	
DELETE RECORD.....	301	TForm.....	273, 275
Deleting		modify collections.....	55
records with TForm.....	279	name.....	57
reports.....	107	organisation.....	61
user.....	223	part.....	21
user group.....	231	record name.....	60
user group member.....	232	record number.....	60
Deletion marker.....	279	record part name.....	61
Delimiter		replace.....	204
TForm.....	273	set restrictions.....	62

type.....	57	<noff>.....	134, 178
DAte .....	69	<nolf> .....	134, 179
description .....	70	<sortfields> .....	134, 193
INteger .....	68	<text variables>.....	134, 196
NUmber.....	68	Forms .....	10
PHrase .....	68	Fragment index	
saving the design .....	70, 71	and TRIP .....	17
Text .....	68	Free text .....	17
Tlme .....	69	Full-text database management system	
types		(TDBS) .....	15, 16
DAte .....	19	Functions	
in TRIP .....	18	reports .....	126, 134
NUmber.....	19	box .....	133
PHrase .....	18	text string .....	126
STring.....	19	<i>FUZz, and the VIF</i> .....	24
TExt.....	18	G marker .....	273, 275
Tlme .....	19	General database properties	
valid values .....	62	creating the database .....	25
Field numbers .....	276	database description .....	29
Field_spec_rec .....	290, 295	database name.....	26
Fields		file locations .....	25
added		modifying database properties .....	31
thesaurus .....	87	advanced properties.....	50, 52
hidden.....	240	files properties.....	34
<b>and data entry</b> .....	241	general properties .....	31
<b>and reports</b> .....	241	indexing properties.....	36
<b>and searching</b> .....	240	physical files.....	27
read-protected .....	240	saving the database .....	30
<b>and data entry</b> .....	241	transaction log .....	27
<b>and reports</b> .....	241	XML enabling the database .....	29
<b>and searching</b> .....	240	General Settings, Limits and Defaults .....	245
FieldSpecRecord .....	290, 295	CCL Command Length Limit .....	245
File		Chinese GBK Character Set.....	245
flat .....	15	Database File size Limit in UNIX .....	246
inverted, and TRIP.....	23	DEfine command defaults .....	246, 247
structures		Euro Currency Symbol	
in TRIP .....	23	Character Set.....	245
File manager.....	220	Searching for.....	245
FIXFIELD.....	300, 302	Open databases limit.....	246
Flat file .....	15	GET LINE .....	299, 302
F-line.....	93	Global updating .....	204
FOR loop		and case sensitivity .....	213
and reports .....	134	and log file .....	214
functions .....	161	copying with .....	212
<append> .....	135	DElete .....	206, 207
<hitlist> .....	135, 164	error checking.....	214
<loop variables> .....	135	examples	
Form		using DElete .....	208, 211
data entry .....	101	using INSert .....	207, 209
Format		using UPDate.....	208, 210
functions		INSert.....	206, 207
<call>.....	134, 148	part records .....	212
<debit> .....	134, 158	<b>records</b> .....	208

targets .....	206	sentence.....	204
type.....	206	subfield.....	204
update		INsert .....	206
domain .....	207	INteger	
target.....	206, 209	restrictions.....	68
type .....	206, 209	Interval	
value.....	206, 209	and values list.....	68
UPDate.....	206, 207	in PHrase pattern .....	64
upper- and lower-case letters.....	213	Italic, as convention .....	11
using a search result.....	208	IX databasename unique ID.log.....	217
using record numbers .....	205	Layout box	
Group		defining a.....	110
user .....	220, 240	Layout retained	
creating .....	229	and TForm.....	274, 276
deleting.....	231	LD databasename unique ID.log .....	217
Group member		Length	
user		marker .....	280
adding .....	232	Letters, as searchable characters .....	40
deleting.....	232	LI databasename unique ID.log .....	217
Groups		Linefeed	
listing .....	234	reports .....	121
Hashed tables.....	23	List	
Head		values and intervals.....	68
field.....	21	Listing	
record .....	21, 22	databases.....	74
Head/part database		groups .....	234
component.....	23	LOAD .....	273
head record .....	22	Loading and Indexing .....	215, 216
part record .....	22	Log file	
record .....	23	and general database properties .....	27
record entity.....	22	naming .....	217
Header.....	241	Logical Names	
reports .....	122	SUPERMAN.....	220
Header_Box		Lower case, as convention .....	11
reports .....	140	Main record .....	21
Hidden fields.....	240	TForm .....	278
<b>and data entry</b> .....	241	Manager	
<b>and reports</b> .....	241	file .....	220, see Database Administrator
<b>and searching</b> .....	240	privileges	
Ignore search characters .....	48	in TRIP .....	20
INCLUDE directory .....	284	responsibility, transferring.....	242
Indent		system.....	20, 220
reports .....	173	user .....	20, 220, 221
Index		Marker	
files, in TRIP .....	23	deletion.....	279
fragment .....	17	field	
INDEX		TForm .....	275
command.....	215	length .....	280
Indexing .....	215	record	
failed batch jobs.....	218	name.....	279
Insert.....	204	name, TForm .....	275
field.....	204	part, TForm .....	275
paragraph .....	204	TForm .....	275

sentence, TForm.....	275	PHrase field	
subfield, TForm.....	275	pattern.....	63
zero		pattern interval.....	64
field .....	279	Positioning	
record .....	279	box .....	113
MESSAGE .....	293, 300	using coordinates .....	113
Meta-record .....	18, 21, 23	using preceding boxes .....	114
M-line.....	93	<b>Print</b>	
N marker .....	275	access .....	243
Name		base access .....	243
user .....	222	<b>group</b> .....	234
Naming conventions .....	12	<b>user</b> .....	234
database.....	26	Privileges	
Natural language text.....	17	database access .....	235
NRX field .....	84	first access form .....	238
NUmber .....	19	Properties	
restrictions .....	68	user .....	224, 225
Numbering		Proportioning	
box .....	113	box	
O-line.....	93	using columns .....	118
Olympic_Games database.....	137	using lines .....	117
structure .....	137	using lines and columns .....	116
Operators, as searchable characters .....	40	PUT LINE .....	299, 302
Output		Q-line.....	93
paged .....	140	R marker.....	273, 275
Output Format Reference Guide .....	144	Read protection .....	240
Output in columns		<b>and data entry</b> .....	241
reports .....	142	<b>and reports</b> .....	241
P marker .....	273, 275	<b>and searching</b> .....	240
Page		Record	
control.....	140	composite.....	21, 23
level		delete .....	204
box .....	140	description of a	
Paged output .....	140	in head/part database.....	23
Paragraph		entity .....	22
defining a.....	44	head .....	21, 22
delete.....	204	main .....	21
insert.....	204	marker	
marker		TForm .....	275
TForm.....	275	meta-.....	18, 21, 23
parse checkbox .....	45	name .....	60
replace.....	204	name field.....	60
separators.....	45	name marker .....	279
start character.....	45	name marker, TForm.....	275
terminators.....	46	name value.....	60
Part		number field .....	60
field.....	21	part.....	21, 22
record .....	21, 22	marker, TForm .....	275
Part record		part name field.....	61
global updating .....	212	part, TForm .....	278
Pattern		unit, in head/part database .....	23
PHrase field .....	63	user .....	223
PHrase.....	18	Records	

copying		<fieldno> .....	129
with global updating.....	212	<fieldtype> .....	129
with TForm .....	280	<FOR> loops.....	161
deleting		<hitlist> .....	135, 164
with TForm .....	279	<hits> .....	127, 166
<b>global updating</b> .....	208	<if-changed> .....	133, 167
in TRIP .....	20	<if-empty> .....	133, 169
part		<if-nonempty> .....	133, 170
global updating.....	212	<if-unchanged> .....	133, 171
replacing with TForm .....	279	<indent> .....	133, 173
updating with TForm .....	279	<link> .....	133, 175
Reindexing.....	217	<loop variables> .....	135
Relational database management system		<noff>.....	134, 178
(RDMS) .....	15	<nolf>.....	134, 179
Replace .....	204	<noorig>.....	133, 180
field.....	204	<numform>.....	127, 182
paragraph .....	204	<occs> .....	127, 184
sentence.....	204	<once>.....	133, 185
string.....	204	<orig>.....	133, 186
subfield .....	204	<pageno>.....	127, 188
Replacing		<paragraphno>.....	129
records with TForm.....	279	<parts>.....	127, 189
Reports .....	105	<rid>.....	127, 190
a description .....	105	<ris> .....	127, 191
and database clusters.....	142	<rname>.....	127, 192
and database Corr.....	110	<sentenceno> .....	129
and FOR loops .....	134	<sortfields> .....	134, 193
and specific field elements.....	112	<subfieldno> .....	129
background text .....	120	<subrid> .....	127, 194
box		<substring> .....	127, 195
functions.....	133	<text variables> .....	134, 196
group .....	119	<timeform>.....	127, 198
size.....	116	<trace>.....	133, 197
carriage return .....	121	<weight> .....	127, 199
copying .....	107	header .....	122
creating.....	107	header_box .....	140
default.....	32	indent .....	173
deleting .....	107	linefeed.....	121
dump .....	130	output in columns .....	142
element.....	106, 120	reserved characters in .....	121
functions .....	126, 134	sample .....	130
<append> .....	135, 144	separator .....	122
<at_end> .....	133, 146	series of spaces or tabs.....	121
<base> .....	127, 147	specification file .....	110
<call>—format.....	134, 148	text functions	
<call>—text string .....	127, 150	field-dependent .....	128
<case> .....	133, 151	text inserts.....	126
<chr> .....	127, 153, 154	text strings	
<curdate> .....	127, 155	field-dependent .....	122
<dateform> .....	127, 156	field-independent.....	125
<debit> .....	134, 158	<i>timestamp</i> .....	112
<ff> .....	127, 159	trailer .....	125
<fieldname>.....	129	trailer_box .....	141



<i>TStamp</i> .....	112	sentence.....	276
Reserved characters.....	121	Series of spaces or tabs	
! symbol.....	121	reports.....	121
/ symbol.....	121	Session index file.....	24
_ symbol.....	121	SET ENTRY.....	299, 300, 302
< symbol.....	121	<b>Show</b>	
> symbol.....	121	access.....	242
Restrictions		base access.....	242
Date.....	69	<b>group</b> .....	234
INteger.....	68	<b>user</b> .....	234
NUmber.....	68	SIF.....	24, 91, 93
Tlme.....	69	Simple box.....	110
to valid values.....	68	S-line.....	93
R-line.....	93	Space character	
S marker.....	273, 275	as convention.....	11
SAMPLES directory.....	287, 288, 290, 297	defining.....	40
SCANASE.C.....	297	Specification file	
Scanit_ase.....	282, 295	reports.....	110
Search form		SQL.....	16
creating.....	201	Start	
Searchable characters.....	40	module, user profile.....	227
and ' symbol.....	40	STatus	
and ! symbol.....	40	and database Alice.....	73
and " symbol.....	40	String	
and # symbol.....	40	delete.....	204
and \$ symbol.....	40	replace.....	204
and & symbol.....	40	STring.....	19
and () symbol.....	40	Structured query language.....	16
and . symbol.....	40	Subfield	
and /		delete.....	204
symbol.....	40	insert.....	204
and ? symbol.....	40	marker	
and + symbol.....	40	TForm.....	275
and character masks.....	40	replace.....	204
and delineators.....	40	SUPERMAN.....	220
and operators.....	40	SWEdish	
and sentence separator defaults.....	40	character folding.....	38
and space character.....	41	symbol.....	11, 44, 48
and truncation symbols.....	40	System	
and word masks.....	40	TRIP basics.....	15
Security		SYSTEM.....	94, 220, 224, 242
database.....	10	System administrator.....	220
Sentence		System logging.....	90
Delete.....	204	activating.....	90
insert.....	204	logicals.....	90
marker		field costs.....	90
TForm.....	275	file format.....	92
replace.....	204	System manager.....	20, 220
separator.....	276	Tab	
separator defaults, as searchable characters		as convention.....	11
.....	40	Table	
Separator		hashed.....	23
reports.....	122	tdbs.conf.....	284, 285

TDBS_		
ACCDIR.....	90	
ACCFLG.....	90, 91, 93, 94	
ASELIBS.....	284, 285	
ERRMAILST.....	218	
EXE/bafini.....	218	
LOG.....	217	
SIF.....	91	
SYS.....	90	
Text		
free.....	17	
natural language.....	17	
Text.....	18	
Text functions		
reports		
field-dependent.....	128	
Text inserts.....	241	
reports.....	126	
Text string		
definition of, in TForm.....	274	
functions		
<base>.....	127, 147	
<call>.....	127, 150	
<chr>.....	127, 153, 154	
<curdate>.....	127, 155	
<dateform>.....	127, 156	
<ff>.....	127, 159	
<fieldname>.....	129	
<fieldno>.....	129	
<fieldtype>.....	129	
<hits>.....	127, 166	
<noff>.....	134	
<numform>.....	127, 182	
<occs>.....	127, 184	
<pageno>.....	127, 188	
<paragraphno>.....	129	
<parts>.....	127, 189	
<rid>.....	127, 190	
<ris>.....	127, 191	
<rname>.....	127, 192	
<sentenceno>.....	129	
<subfieldno>.....	129	
<subrid>.....	127, 194	
<substring>.....	127, 195	
<timeform>.....	127, 198	
<weight>.....	127, 199	
reports.....	126	
Text strings		
and ! symbol.....	121	
and / symbol.....	121	
and _ symbol.....	121	
and < symbol.....	121	
and > symbol.....	121	
and TForm.....	274	
reports		
field-dependent.....	122	
field-independent.....	125	
TEXTASE.C.....	288	
TFOFIELD.C.....	290	
TForm		
and <CR><LF>.....	274, 275, 276	
and <SP>.....	274	
and control master.....	274	
and control strings.....	273	
and layout retained.....	274, 276	
and text strings.....	274	
and the BAF.....	279	
copying records with.....	280	
deleting records with.....	279	
deletion marker.....	279	
F marker.....	273, 275	
field marker.....	273, 275	
G marker.....	273, 275	
main record.....	278	
N marker.....	275	
P marker.....	273, 275	
paragraph marker.....	275	
paragraph/subfield marker.....	273, 274	
paragraphs and sentences in.....	276	
R marker.....	273, 275	
record marker.....	273, 275	
record name marker.....	275	
record part.....	278	
record part marker.....	273, 275	
replacing records with.....	279	
S marker.....	273, 275	
sample file.....	277	
sentence marker.....	273, 274, 275	
subfield marker.....	275	
text string, definition of.....	274	
updating records with.....	279	
zero field marker.....	279	
zero record marker.....	279	
Thesali database.....	10	
Thesauri.....	80	
what is.....	80	
Thesaurus		
added fields.....	87	
creating.....	83	
data layout.....	84	
design		
character sets.....	87	
defaults.....	87	
description.....	87	
field definition.....	87	
general properties.....	87	



other properties .....	87	value .....	206, 209
special fields .....	87	UPDate .....	206, 207
design .....	87	using a search result .....	208
example .....	81	using record numbers .....	205
filling .....	88	Upper case, as convention .....	11
data entry .....	88	User .....	
TForm .....	88	administration .....	220
structure .....	84	creating .....	222
top terms .....	83	deleting a .....	223
Time .....	19	end .....	221
restrictions .....	69	group .....	220, 240
Timestamp .....		creating .....	229
reports .....	112	deleting .....	231
Top terms .....		member, adding .....	232
thesaurus .....	83	member, deleting .....	232
Trailer .....		individual .....	220, 221
reports .....	125	name .....	222
Trailer_Box .....		password .....	222
reports .....	141	<b>print</b> .....	234
Trigram .....	24	<b>print user group</b> .....	234
TRIP .....		properties .....	224
and inverted file organization .....	23	record .....	223
database basics .....	20	responsibility, transferring .....	233
index files in .....	23	<b>show</b> .....	234
jump table .....	285	<b>show user group</b> .....	234
manager privileges .....	20	User group .....	220
naming conventions .....	12	User manager .....	20, 220, 221
records in .....	20	User profile .....	
system basics .....	15	date form separator characters .....	226
system data dictionary .....	see CONTROL	start module .....	227
TRIPmanager navigation .....	14	User properties .....	225
Truncation symbols, as searchable characters .....	40	company information .....	227
TStamp .....		full name .....	225
reports .....	112	groups list .....	228
U-line .....	93	ignore TRIP password .....	225
Unigram .....	24	login procedure .....	227
UPDate .....	206	privileges .....	226
Updating .....		procedures list .....	227
global .....	204	rights list .....	229
and case sensitivity .....	213	session parameters .....	227
and log file .....	214	start module .....	227
copying with .....	212	Values and intervals list .....	68
DELeTe .....	206, 207	VIF .....	23, 204
error checking .....	214	and general database properties .....	27
INSert .....	206, 207	Vocabulary index file .....	23
part records .....	212	Vocabulary Index File .....	see also VIF
targets .....	206	Word .....	
type .....	206	masks, as searchable characters .....	40
update .....		WRITE MESSAGE .....	299
domain .....	207	Zero .....	
target .....	209	field marker .....	279
type .....	209	record marker .....	279

