



digital
vision
group

TRIP Administration with TRIPclassic

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.



About This Guide	8
The TRIP Documentation Library	8
The Structure	8
Conventions Used in this Guide	9
CCL and the TRIPclassic Numeric Keypad	10
TRIP Naming Conventions	11
Part 1: Database Administration	12
Chapter 1: Fundamentals	13
TRIP System Basics	13
Introduction	13
Data Models	13
Data Organisation	15
TRIP Field Types	16
The CONTROL Database	17
TRIP Manager Privileges	18
TRIP Database Basics	19
Records	19
File Structures	21
Chapter 2: Databases	23
Navigation (Screen Forms)	23
General Database Properties	24
Physical Files	25
Special Database Fields	27
Defaults	29
Character Sets	31
Description of the Database	34
Overlay Forms for General Database Properties	34
Field Definition	44
Screen Navigation	44
Field Attributes and Restrictions	47
Overlay Forms For Field Definition	58
Saving the Database Design	59
Modifying the Design	60
Deleting a Design	62
Copying a Design	63
Related CCL Commands	64
SStatus	64
Show	64
Print	65
IMPORT and EXPORT	65
Database Clusters	66
Creating a Cluster	66
Modifying a Cluster	67
Deleting a Cluster	67
Related CCL Commands	69
Single Namespace	69
Chapter 3: Thesauri	70
What Is a Thesaurus?	70
A Simple Thesaurus	71
Creating a Thesaurus	73
Thesaurus Design	75
Data Layout	75
Database Design	78

Filling The Thesaurus.....	82
Using TForm	82
Using Data Entry	82
Related CCL Commands	83
STatus	83
Show.....	83
IMPOrt/EXPOrt.....	83
Chapter 4: System Logging Functions.....	84
Assigning Field Costs.....	84
Activating System Logging Functions.....	84
Logical Names	84
File Format.....	87
Part 2: Forms	90
Chapter 5: Data Entry Forms.....	91
Screen Navigation.....	91
Entry Form Components	92
Creating a Data Entry Form	92
Fixed Screen Text.....	94
Defining a Field	95
Form-Based ASEs	105
Field-Specific ASE Overlay	106
Moving Between Pages.....	106
Cutting and Pasting.....	106
Drawing Borders	107
Tab Order.....	107
Saving the Form.....	108
Copying and Deleting Entry Forms.....	108
Related CCL Commands	109
Show.....	109
Print	109
EXPOrt/IMPOrt.....	109
Chapter 6: Output Formats	110
The Format	110
Creating an Output Format.....	112
Defining Layout Boxes	113
Background Text.....	123
Functions	129
Page Control	143
Output Formats for Database Clusters	145
Copying and Deleting Formats	145
Related CCL Commands	146
Output Format Reference Guide	147
<APPEND>	147
<AT_BEGIN>	148
<AT_END>	149
<BASE>	150
<CALL> – Format.....	151
<CALL> – Text String.....	153
<CASE>	155
<CHARSET>.....	157
<CHR>	158
<CLASS>	159
<CURDATE>	160

<DATEFORM>	161
<DEBIT>	163
<ENTITIFY>	164
<FF>	165
<FOR> Loops	167
<HITLIST>	170
<HITS>	172
<IF-CHANGED>	173
<IF-EMPTY>	175
<IF-NONEMPTY>	176
<IF-UNCHANGED>	177
<INDENT>	179
<LINK>	181
<NOFF>	184
<NOLF>	185
<NOORIG>	186
<NUMFORM>	188
<OCCS>	190
<ONCE>	191
<ORIG>	192
<PAGENO>	194
<PARTS>	195
<RID>	196
<RIS>	197
<RNAME>	198
<SORTFIELDS>	199
<SUBRID>	200
<SUBSTRING>	201
<Text Variables>	203
<TRACE>	204
<TIMEFORM>	205
<WEIGHT>	207
Chapter 7: Search Forms	208
Elements of a Search Form	208
Anatomy of a Search Form	209
Creating Search Forms	213
Search Form Definition	215
Search Box Layout and Comments	217
Search Box Specification and Search Logic	219
Command Box Specification	223
Command Variables	226
Form Errors	227
Part 3: Batch Update	228
Chapter 8: Global Updating	229
Command Overview	229
Updating Using Record Numbers	230
Updating Using a Search Result	233
Global Updating of Part Records	237
Copying With Global Update	237
Case Sensitivity	238
The Log File	239
Error Checking	239
Chapter 9: Loading, Indexing and Reindexing	240

Index.....	240
Load/Index	242
Load.....	242
Checking the Results	243
Error Logging	243
Reindexing a Database.....	244
When Batch Jobs Fail	244
On UNIX and Windows systems	244
On UNIX systems only	244
Part 4: Database Security.....	245
Chapter 10: User Privilege.....	246
Access Privileges.....	246
The System Manager.....	246
The TRIP 'Superman' Logical Name	246
The File and User Managers	246
The User Group	246
The Individual or End User.....	247
Creating a New User.....	247
Deleting a User	248
User Profiles	249
Transferring User Responsibility	251
Creating a User Group	251
Deleting a User Group.....	252
Adding a Group Member	253
Deleting a Group Member	253
System Manager Privileges.....	253
Setting a New Password	254
Related CCL Commands	255
Show.....	255
Print	255
Chapter 11: Access Rights	256
Database Access Rights Definition	258
General Field Access	258
Field-Level Access	259
Record-Level Access	260
The Hierarchy of Access Rights	261
Database Cluster Access	262
About Read-Protected Fields	262
Transferring Database Ownership.....	263
Related CCL Commands	264
Show.....	264
Print	264
Part 5: Appendix and Index	265
Appendix A.....	266
General Settings, Limits and Defaults	266
Support for the Euro Currency Symbol.....	266
Searching for the Euro symbol	266
Support for the Chinese character set GBK.....	266
Limit to TRIPclassic CCL Command Length.....	266
No Limits to Database and Index File Sizes	267
Limit to the Number of Open Databases	267
Defaults for the DEfine command.....	267
TRIPserver Crash Handling (Windows only)	268

Appendix B.....	269
Keyboard Key Combinations for Emulating VT Keypad Keys	269
Emulate <PF1>, <PF2>, <PF3> and <PF4>:.....	269
Emulate the keypad number keys:	269
Emulate the keypad keys, < . >, < , > and < - >	269
Emulate the keys <Page Up>, <Page Down>, <Backspace> and keypad <Enter>	269
Obtaining Version and License Information	270
TRIPsystem Version Information	270
Updating a TRIP Product License Key	270
TRIP User Account Validation Methods	271
Overview	271
LDAP	271
Local System Validation	273
TRIP Standalone Usernames	274
TRIP Grids	275
Introduction to TRIP grid computing	275
Classification Schemes	277
Introduction to Classification Schemes	277
Scope Search Facility	278
The new Scope Search facility	278
Scope Search Example	278
Appendix C:.....	281
TRIP Programming	281
TForm	281
Control Strings	281
Text Strings	282
Record, Record Part, Field and Subfield Markers.....	282
Adding Records With TForm	284
Updating Records With TForm	287
Data Type STring and the Length Marker.....	288
Copying Records Using Print TForm	288
Application Software Exits (ASEs).....	289
Summary.....	289
The Format of an ASE Routine	292
Linking ASE Routines to TRIP.....	292
Debugging ASE routines	294
CCL ASEs.....	295
Output Format ASEs	295
TForm Load ASEs.....	298
Index ASEs	303
Data Entry ASEs (TRIPclassic only).....	307
Search Form ASEs (TRIPclassic only)	311
TRIPsystem Callback Functions for ASE Routines.....	311
TRIPclassic Callback Functions for ASE Routines	311
TRIP API Reference Guide	311
List of Figures and Tables	312
Index	317

About This Guide

The TRIP Documentation Library

This manual describes the administration of TRIPsystem via the TRIPclassic interface, which encompasses the creation and maintenance of databases and the management of user access to the system and its databases. Other members of the TRIP documentation library include the TRIPsystem Installation Guide, Release Notes, TRIPclassic User Guide, TRIPmanager Administration Guide, TRIPmanager User Guide, TRIP CCL Command Reference and the TRIPsystem Application Programming Reference.

If you are not already familiar with searching and CCL (TRIP's Common Command Language), we recommend that you read the *TRIPclassic 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 *TRIPclassic CCL Command Reference*.

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

The Structure

This manual is divided into six main parts, Database Administration, Forms, Batch Update, Database Security and Appendix:

- | | |
|-----------|--|
| 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 the creation, maintenance and usage of data entry forms, output formats 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: | Appendix, TFORM (Trip output FORMat) and ASEs (Application Software Exits). |

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

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 <i>fieldtype</i> or <i>databasename</i> , and to emphasize important terms and concepts
bold	used to indicate anything that TRIP recognizes or can interpret and act upon, such as the things mentioned above (<Tab>, CCL Search , DEfine , Alice , and wonderland)
lower case	used for terms and variables where variables are also italic
Upper Case	used for proper names such as the database ALICE
Courier	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>
↵	Reversed 'L' arrow used after commands, meaning 'press either <Enter> or <Return>'
<Next>	the <Page Down> or <Next Page> key
<Prev>	the <Page Up> or <Previous Page> key
<Gold>	the <PF1> key
<Leave>	the <PF3> key
< >	space character
<CR>	carriage return
<LF>	line feed
<NL>	new line
<FF>	form feed
<VT>	vertical tab
<kp>	a key on the numeric keypad
" "	messages from TRIP

CCL and the TRIPclassic Numeric Keypad

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

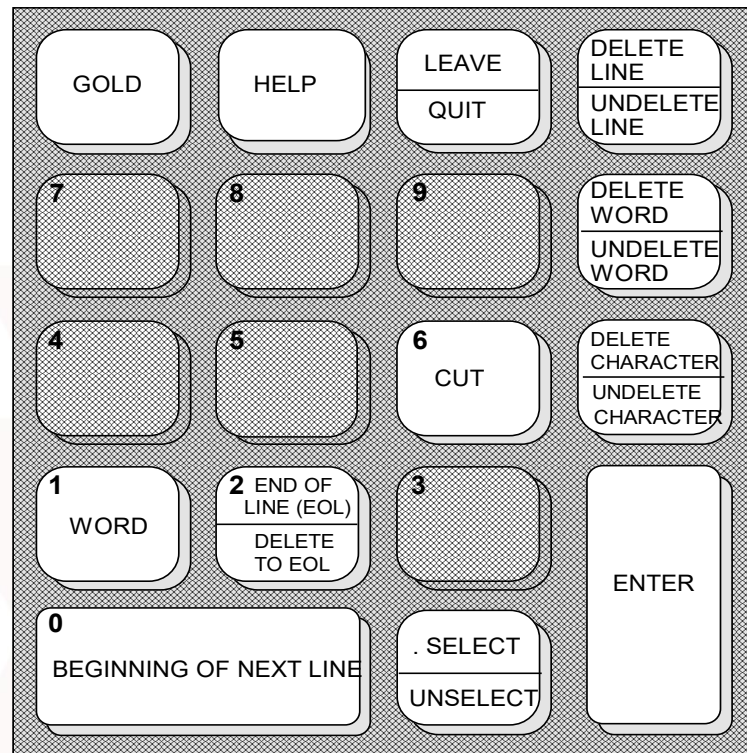


Figure 1 – TRIPclassic VT Numeric Keypad

Note:

- The above keypad is for a true VT terminal keyboard, for UNIX when using a terminal window with a standard PC keyboard keypad and also on a standard Windows keyboard keypad when using *TRIPclassic* for Windows.
- When using a standard 102 key keyboard, the VT keypad function keys **<PF1>** to **<PF4>** are also transposed to the PC main function keys **<F1>** to **<F4>** respectively.
- For VT terminal emulator key combinations, see *Appendix B: Keyboard Key Combinations for Emulating VT Keypad Keys* on page 269 of this guide for more details.

TRIP Naming Conventions

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

Category	Content Alphanumeric?	Length ¹	First Letter Alphabetic?	Allowable Punctuation
File	Yes	128	Yes	Underscore
Database	Yes	16 (64)	Yes	Underscore
Field	Yes	16 (64)	No	Underscore
Procedure	Yes	16 (64)	No	Underscore
Output Format	Yes	16 (64)	No	Underscore
Entry Form	Yes	16 (64)	No	Underscore
Search Form	Yes	16 (64)	No	Underscore
Group	Yes	32 (128)	No	Underscore
User	Yes	32 (128)	No	Underscore
Password	Yes	32 (128)	No	Underscore

Table 0–1 TRIP naming conventions

¹ maximum length in characters (including file paths where applicable). The values in parentheses apply to programs using TRIP 8.1 or later with the database design APIs as introduced in TRIP 8.0. These include TRIPmanager and applications written using TRIPnpx or TRIPjxp. The longer names are not currently supported in TRIPclassic and, even if found to be working in some cases, they must not be relied upon for TRIPclassic production use.

Part 1:

Database Administration



Chapter 1: Fundamentals

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.

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 useful 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 natural language text, such as books or documents, rather than on small, discreet information systems which are more suited to relational database management systems.

The *fragment index* is a unique feature of the TRIP database system, which provides significant performance increases when searching for truncated terms. For instance, in the previous table the fragment index is used to locate

terms, which in turn are used to locate content within the database itself. The example:


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\$

yields 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 are typical of *free text* applications. Free text is used here to mean natural language text, as found in books, letters, reports 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 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 will 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**, **PHrase**, **DAte**, **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.

PHrase 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 **PHrase** field, but each is restricted to 255 characters or fewer.

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 with all subfield contents as well as each individual word is noted in a TRIP 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.

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, output formats, 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.

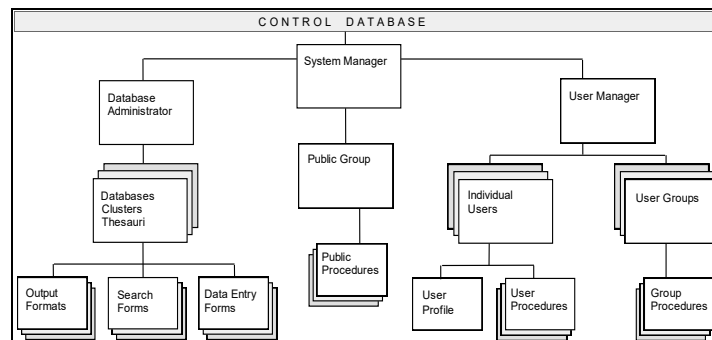


Figure 1-2 The CONTROL database

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 output formats, 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.

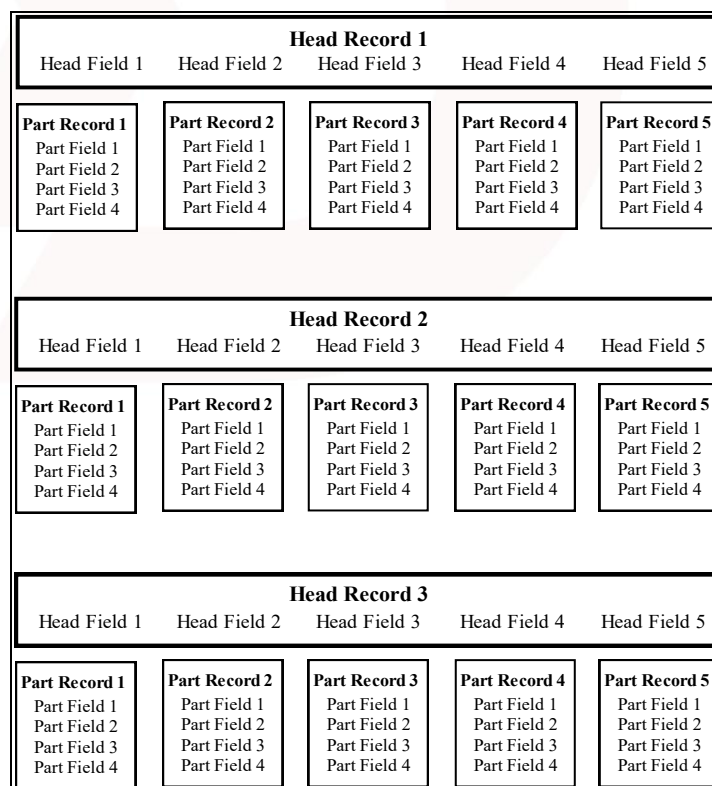


Figure 1–3 Head and part records in a database

Using the demonstration database **Carroll** as an example, head records made up of head fields now contain all of the *chapter information* contained in

Head Record 1				
Head Field 1	Head Field 2	Head Field 3	Head Field 4	Head Field 5
Part Record 1 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 2 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 3 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 4 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 5 Part Field 1 Part Field 2 Part Field 3 Part Field 4

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			
Book	Chapter	Chaptnr	Person
Part Record 1 Speaker Txt Verse Txt2	Part Record 2 Speaker Txt Verse Txt2	Part Record 3 Speaker Txt Verse Txt2	Part Record 4 Speaker Txt Verse Txt2
Part Record 5 Speaker Txt Verse Txt2			
Head Record 2			
Book	Chapter	Chaptnr	Person
Part Record 1 Speaker Txt Verse Txt2	Part Record 2 Speaker Txt Verse Txt2	Part Record 3 Speaker Txt Verse Txt2	Part Record 4 Speaker Txt Verse Txt2
Part Record 5 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.

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

Figure 1–5 A head record

2. 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
Part Record 1 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 2 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 3 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 4 Part Field 1 Part Field 2 Part Field 3 Part Field 4	Part Record 5 Part Field 1 Part Field 2 Part Field 3 Part Field 4

Figure 1–6 A part record

3. 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
Part Record 1	Part Record 2	Part Record 3	Part Record 4	Part Record 5
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 1–7 A record entity

4. 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
Part Record 1	Part Record 2	Part Record 3	Part Record 4	Part Record 5
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

5. 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
Part Record 1	Part Record 2	Part Record 3	Part Record 4	Part Record 5
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** (**BA**se **F**ile), **BIF** (**BA**se **I**ndex **F**ile) and **VIF** (**V**ocabulary **I**ndex **F**ile).

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 BAsE 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 TRIPclassic User Guide and TRIPclassic 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 a disconnect, such as that created by pressing <Ctrl><Y> or a line drop during a modem session. This includes search and print order histories, search language (English, Swedish etc.), open thesauri and any maximums, minimums or mapping that have been defined.

Chapter 2: Databases

Before beginning, be sure your system manager has assigned you the right to create databases (database administrator privileges).

Navigation (Screen Forms)

After logging on to TRIP, the first screen to appear is the Primary Option Menu.

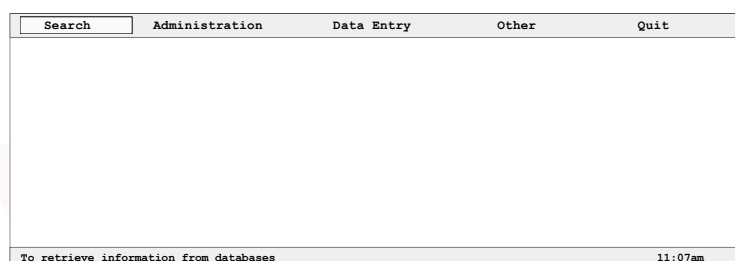


Figure 2-10The Primary Option Menu

The cursor will be positioned in the upper left corner of the menu, with **Search** bolded or highlighted.

The pathway for database creation from the Primary Option Menu looks like this:

```

Primary Option Menu:
Administration ↵
    Databases ↵
        DB Design ↵
            Create/Modify Database ↵
                Create/Modify Database Design
Form ↵
    
```

From **Search**, move the cursor right one cell using the <→ **Arrow**> key, and press <**Return**> or <**Enter**> (symbol: ↵) or the <↓ **Arrow**> key. The *Administration* drop-down menu appears, with *Databases* as the default cursor position.

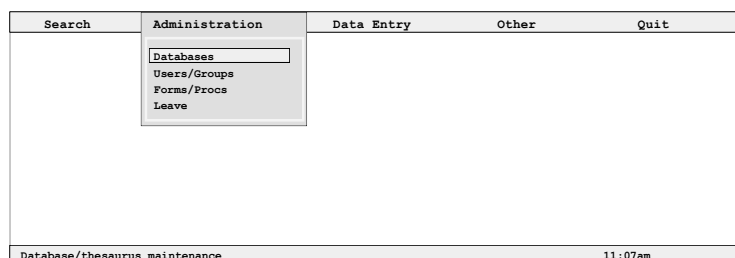


Figure 2-11The Administration menu

If you wish to return to the Primary Option Menu at this point, press <**Leave**>, or <**Gold**><**Leave**> to quit TRIP altogether.

Press ↵ to choose the *Databases* option, which leads to a new menu headed by the cursor default *DB Design*.




Figure 2–12The DB Design option

Press **↓** or **<↓ Arrow>** for the *DB Design* option list, the *Create/Modify* menu. Again, pressing **<Leave>** will return you to the Primary Option Menu; **<Gold><Leave>** exits TRIP.

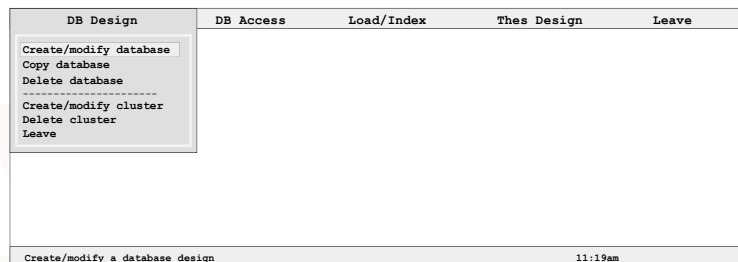


Figure 2–13The Create/Modify Database option

Select the cursor default alternative *Create/Modify* with **↓** to display the forms for designing a new database.

General Database Properties

The first form you will need to fill asks for the name of the database you would like to create.

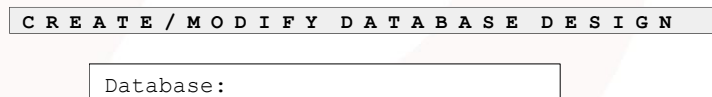


Figure 2–14The Create/Modify Database Design form

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 underscore (_).

Type an appropriate (preferably descriptive) name beside 'Database:' and press **↓**. The *General Database Properties* form appears.

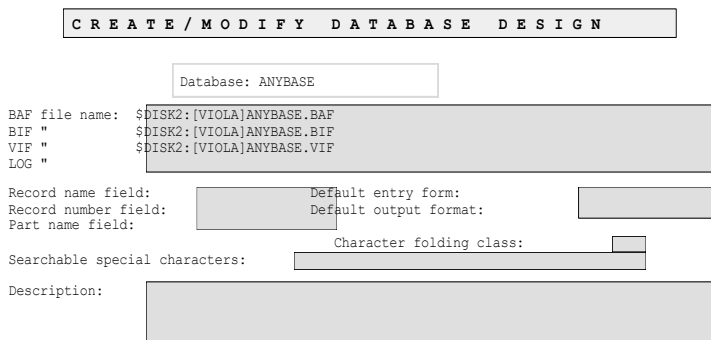


Figure 2–15The General Database Properties form

The *General Database Properties* form has several *overlays*, or associated forms that can be accessed only from *General Properties*. These are presented in the table below, with the keystrokes required to access them.

General Database Properties Overlay Form	Press Key to Access
ASE	<kp 1>
Sentences and Paragraphs	<kp 2>
Index/Update Submission	<kp 3>
Field Definitions	<kp 7>

Table 2–5 General database properties overlay forms

Use the <↓ **Arrow**>, <Tab> or <Return> keys to advance from field to field, and <↑ **Arrow**> to reverse direction and move upscreen. The <Enter> key saves your database design.

The numeric keypad contains the keys needed for the *General Database Properties* form.

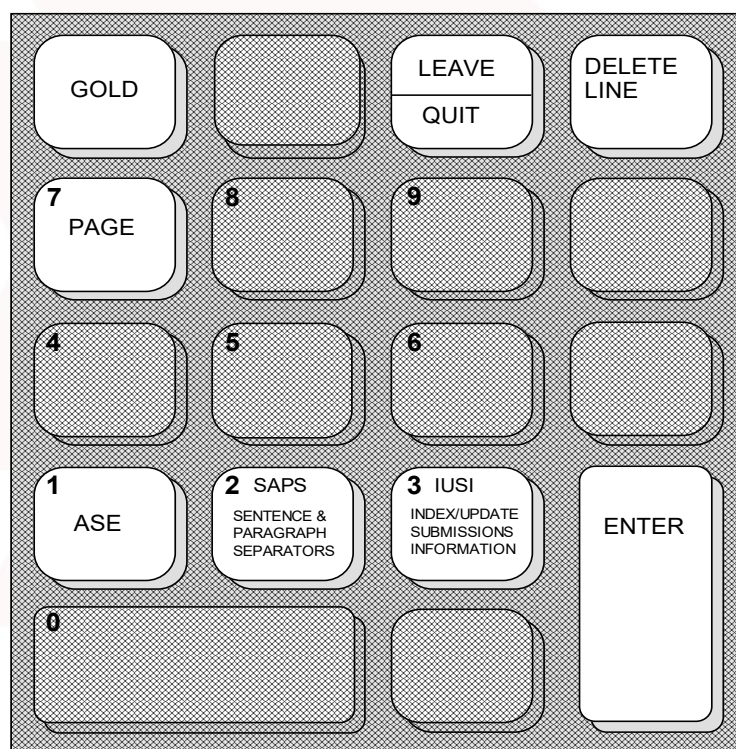


Figure 2–16 The General Database Properties design keys

Each part of the *General Database Properties* form is presented individually in the sections that follow.

Physical Files

This portion of the form allows you to specify the location of the database files **BAF**, **BIF** and **VIF** (and **LOG** if desired) within your file system. TRIP creates the physical database files when these specifications are saved, and the default file location (unless otherwise indicated) is the current directory..

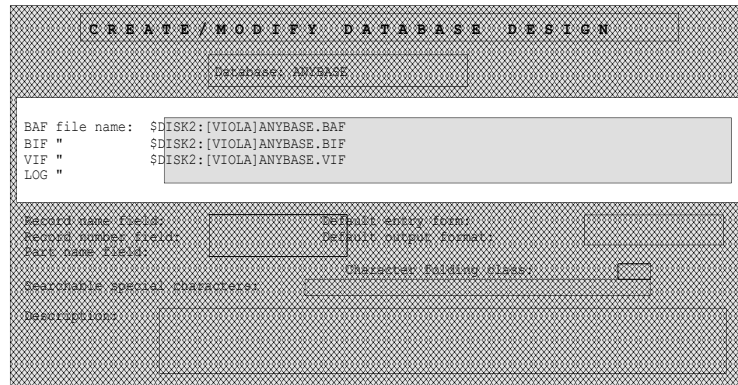


Figure 2–17 The Physical File fields

You may wish to exchange the device/directory specification for a *logical name* or *environment variable* to facilitate the use and administration of the database. We suggest using a naming convention such as:

application *name*_BASES

For example, if your application is called MYAPP, the respective file names would be:

UNIX: \$MYAPP_BASES/XYZ.BAF
 \$MYAPP_BASES/XYZ.BIF etc.

Windows: MYAPP_BASES:XYZ.BAF
 MYAPP_BASES:XYZ.BIF etc.

So, in UNIX, the command formats are:

in CSH,

setenv variable equivalence

for example:

setenv MYAPP_BASES /usr/app/myapp/bases ↵

in BSH and KSH,

variable = equivalence

for example:

**MYAPP_BASES=/usr/app/myapp/bases ;
export MYAPP_BASES ↵**

In Windows, it is recommended that any such TRIP environment variables are created in the [NonPrivileged] section of the *tdbs.conf* file, which is located in the *conf* directory of the TRIPsystem installation:

MYAPP_BASES=C:\Tieto\Bases

Note:

*Using the *tdbs.conf* file is the recommended method for TRIP environment variable creation.*

Transaction Log

You can attach a **TForm** (TRIP's batch upload **T**ext file **FORM**at) log file to the database via the *Log File Name* field on this form. This will record 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.

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 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)
```

and in UNIX,

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

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.*

Special Database Fields

These include the record name, record number and record part name fields.

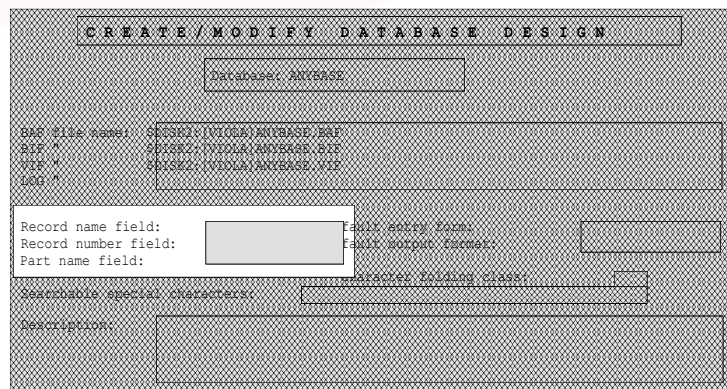


Figure 2–18 The Special Database fields

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 2–6 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 ↵
```

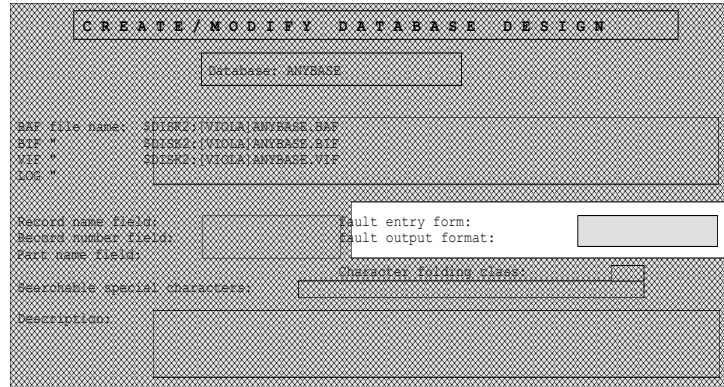
was not available.

Record Part Name Field

If your database design employs head-part record structure, you should always utilize record part name fields, which allow the unique identification of parts within a composite record. If you do not, database users will be unable to take advantage of the selective part editing features TRIP provides, such as **Cut**, **Paste**, **Select** and **Insert**. Refer to the section entitled 'Data Entry With Record Parts' in Chapter Nine of the *TRIPclassic User Guide* for more information on the editing of record parts.

Defaults

These include data entry forms and output formats.



The screenshot shows a window titled "CREATE/MODIFY DATABASE DESIGN". It contains several input fields and labels:

- Database:** A text box containing "ANYBASE".
- SAF file name:** A text box containing "S:\SAFE2\VTOLA\ANYBASE.SAF".
- SIP #:** A text box containing "S:\SAFE2\VTOLA\ANYBASE.SIP".
- VIP #:** A text box containing "S:\SAFE2\VTOLA\ANYBASE.VIP".
- LOG #:** A text box containing "S:\SAFE2\VTOLA\ANYBASE.LOG".
- Record name field:** A text box.
- Record number field:** A text box.
- Part name field:** A text box.
- Default entry form:** A text box.
- Default output format:** A text box.
- Searchable special characters:** A text box.
- Character folding case:** A text box.
- Description:** A large text area.

Figure 2–19 The Default Format fields

Entry Form

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.

A user opening Data Entry from the Primary Option Menu follows this pathway:

```
Primary Option Menu:
Data Entry ↵
  Add ↵
    Add Records To Database Form ↵
      Database Name:
      [Data Entry] Form: ↵
```

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.

Output Format

If you do not provide the name of a default output format, the system will show all output using its built-in default format 'Dump'. This format 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**:

```
Record 1  in Database ALICE
CHAPTER: Down the Rabbit Hole
PERSON : Alice's sister, White Rabbit
CHAPTNR: 1
TXT    : 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.

Record 2  in Database ALICE
CHAPTER: Down the Rabbit-hole
PERSON : White Rabbit
..>>>
```

(Enter your command, please.)
(Database)

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit
11:39am

Figure 2–20Sample SYSTEM default output

Designating a default output format 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 tweedled
DEfine Format=outputformatname ↵
show ↵
```

a user may simply enter

```
BASE alice
Find tweedled
show ↵
```

after a search.

The **Define Format** statement is not needed if an output format has been specified.

Character Sets

This includes character folding specifications and the definition of searchable special characters.

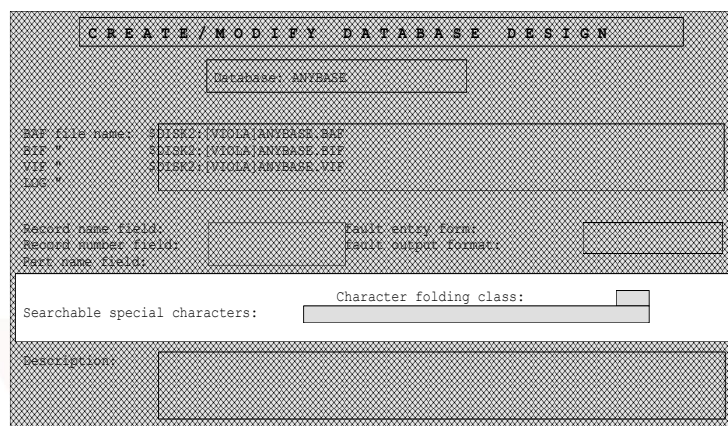


Figure 2–21 The Character Set fields

Character Folding

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					œ			
	Ç							
						ß		

Table 2–7 Special characters

The *Character Folding Class* design field 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 MUL (MULTinational), ENG (ENGLISH) and SWE (SWEdish).

The default is MUL, in which no character folding is done, e.g. an é is a singular character separate from e, ã is not indexed as a, and so forth.

The second folding class, ENG, 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 third class, SWE, is identical to ENG except that å, ä, and ö are recognized as singular characters.

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

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

Table 2–8 The character folding classes

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.

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. 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 2–9 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 a **DEFINE SPACE** order.

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 *TRIPclassic CCL Command Reference* for further information regarding **DEFINE SPACE**, truncation and masking.

Description of the Database

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

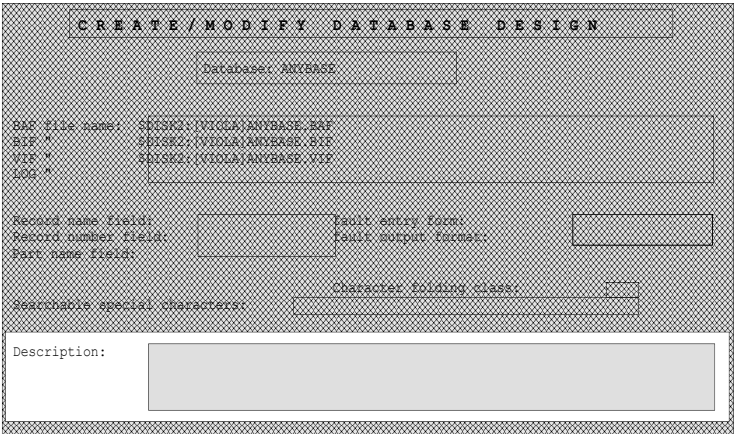


Figure 2–22 The Database Description field

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 user requests to view the **Status** for that database.

Overlay Forms for General Database Properties

Several screens, called *overlays*, are superimposed on the *General Database Properties* form. These include *ASE*, *Sentences and Paragraphs*, *Index/Update* and *Field Definitions* outlines, and are accessible using various **<kp>** (numeric keypad) keys, as shown in the table below:

General Database Properties Overlay Form	Press Key to Access
ASE	<kp 1>
Sentences and Paragraphs	<kp 2>
Index/Update Submission	<kp 3>
Field Definition	<kp 7> or <Next Page>

Table 2–10 General database properties overlay forms

ASEs (Application Software Exits)

This form defines ASEs for **TForm** loading of records. Press **<kp 1>** from the *General Database Properties* design form to open it.

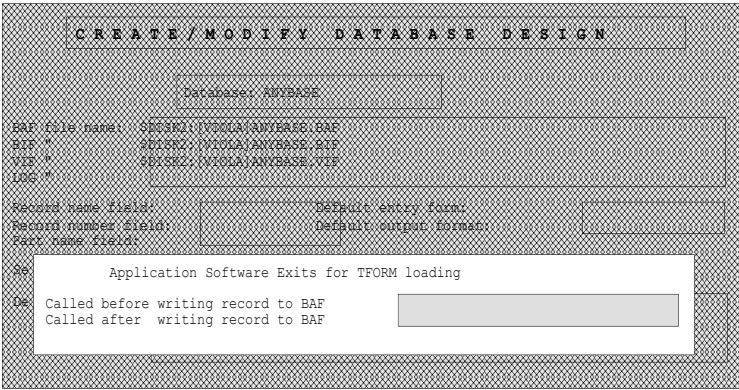


Figure 2–23 The ASE overlay

You may enter the names of any ASEs you wish to call either before or after writing to the **BAF**. Press the **<↓ Arrow>**, **<↑ Arrow>**, **<Tab>** or **<Return>** keys to toggle between the two fields. When finished, press **↓** to return to the *General Database Properties* form.

See the Appendix of this manual for more information regarding ASEs.

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. Open this definition form by pressing **<kp 2>** from the *General Database Properties* form.

To return to the *General Database Properties* form without saving changes, press **<Leave>**.

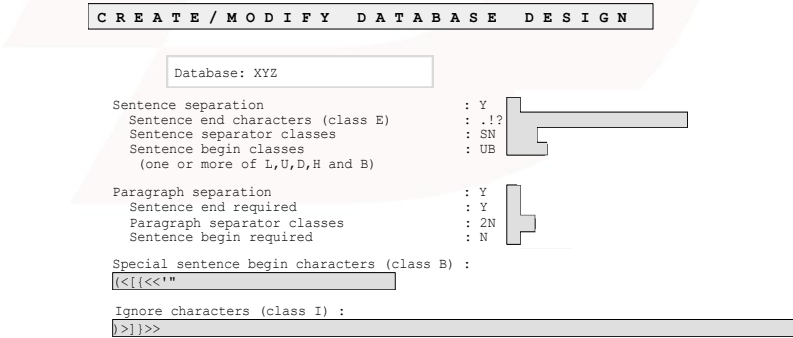


Figure 2–24 The Sentence/Paragraph Separation overlay

The system defaults for each category are presented onscreen as seen above.

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 2–11 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).

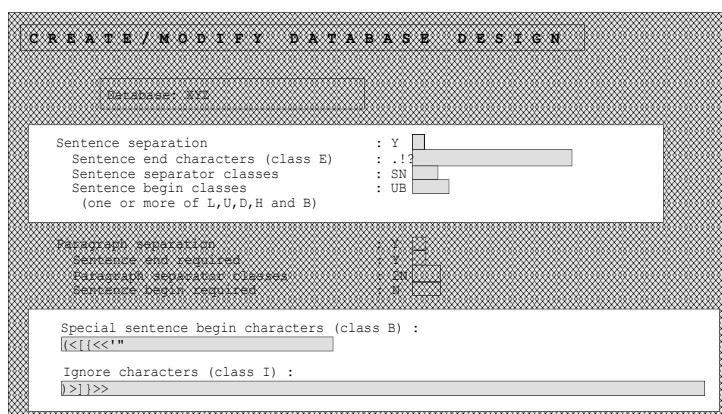
With the exception of the <CR> (Carriage Return), which is permanently assigned to Class **I**, Class **S** contains the space character (ASCII value 32) as well as all control characters (ASCII values 0-31) not belonging to Class **N**.

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

These fields allow you to define the beginning, the end and the separations between sentences.



CREATE / MODIFY DATABASE DESIGN

Database: XYZ

Sentence separation : Y ☐

Sentence end characters (class E) : .!?" ☐

Sentence separator classes : SN ☐

Sentence begin classes : UB ☐

(one or more of L,U,D,H and B)

Paragraph separation : Y ☐

Sentence end required : N ☐

Paragraph separator classes : SN ☐

Sentence begin required : N ☐

Special sentence begin characters (class B) :
[[<[<<["

Ignore characters (class I) :
]>]]>>

Figure 2-25 The Sentence Separation fields

Sentence Separation fields

The default values on the database design definition form are shown in Figures 2-15 and 2-16. A more detailed description of these conditions follows.

Sentence Separation

The options for sentence separation are Y (Yes) and N (No). TRIP defaults to Yes, separating text into paragraphs using its default sentence definition if none is provided by the database designer.

If you choose Yes, 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).

Sentence End Characters (Class E):

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 **!?**

Sentence Separator Classes

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.

Sentence Begin Classes

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**.

User-Defined Classes

Special Sentence Begin Characters (Class B)

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 **(< [{ « ' "**

Ignore Characters (Class I)

This option specifies which character(s) will belong to Class **I**. 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 2–12 Sentence definition in TRIP

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

Defining a Paragraph

Paragraph Separation Fields

Paragraph separation

Paragraph separation is independent of sentence separation, with options Y for Yes, N for No, and a default of Yes.

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.

Sentence End Required

The options are Y for Yes and N 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.

Paragraph Separator Classes

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 **N** 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.

Sentence Begin Required

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 2–13 Paragraph definition in TRIP

The illustration above shows two paragraphs and their separators.

Considerations

- 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.
- 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.
- 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.

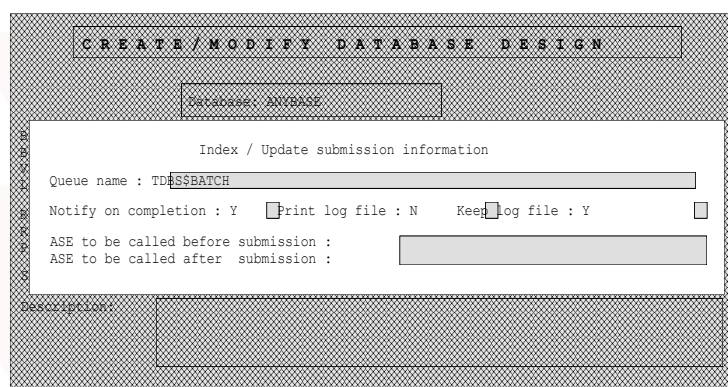
- Be careful when changing the default settings for a pre-existing database. You must extract all data before the changes are made and reload it into the new design.

Index/Update Submission (Queue Management)

Note:

While this feature is visible in TRIPclassic on other operating systems, Queue Management is functional only as a legacy item for the VMS operating system.

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. Open this form by pressing **<kp 3>** from the first page of the database design form. **<Leave>** returns you to the *General Database Properties* form without saving.



The screenshot shows a window titled 'CREATE / MODIFY DATABASE DESIGN'. Inside, there's a section titled 'Index / Update submission information'. The 'Database' field is set to 'ANYBASE'. The 'Queue name' field contains 'TDBS_BATCH'. Below this, there are three checkboxes: 'Notify on completion : Y' (checked), 'Print log file : N' (unchecked), and 'Keep log file : Y' (unchecked). At the bottom, there are two empty text fields labeled 'ASE to be called before submission :' and 'ASE to be called after submission :'. A 'Description:' label is at the bottom left of the form area.

Figure 2–26 The Index/Update Submission overlay

Queue Name

This option allows for a particular batch queue to be specified for the database load, global update and index jobs. You must use either a valid queue name or logical name, such as the default TDBS_BATCH.

Notify On Completion

This option can be set to notify the user that a TRIP batch job has completed, providing the user's broadcasting has not been switched off. Valid values are Y (Yes) and N (No); the default setting is Y.

Print Log File

This option specifies whether or not the Log file generated by the TRIP batch job should be printed to the default printer defined for the user. Y (Yes) and N (No) are the Print Log file options accepted; the default setting is N.

Keep Log File

This option can be set to store the Log file generated by a TRIP batch job, either in the user's default directory or the directory indicated by the logical name TDBS_LOG (if defined). Y (Yes) and N (No) are the only values accepted, and the default setting is Y.

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.



Field Definition

To open the *Field Definition* form, press the **<Next Page>** or **<kp 7>** key from *General Database Properties*. Here you may specify a name, type, index mode, content layout, composite record membership, presence and number of subfields and content validation schemes for each field as applicable.

The *Field Definition* form has several *overlays*, or associated forms that can be opened only from *Field Definitions*. These are presented in the table below, with the keystrokes required to access them.

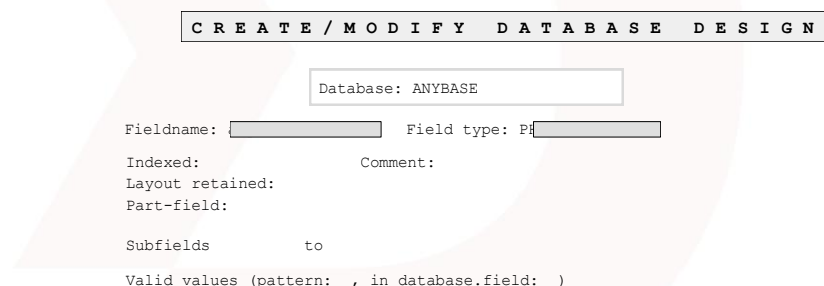
Field Definition Overlay Form	Press Key to Access
Field List	<kp 9>
ASE	<kp 1>
Accounting	<kp 7>

Table 2–14 Field definition overlay forms

Screen Navigation

Use **<Tab>**, **<→ Arrow>**, **<↓ Arrow>** or **↓** to move between design fields. Each screen holds a single field definition.

Press **<Enter>** after completing each screen to save, and **<Leave>** when finished. **<Gold><Enter>** clears the form without saving the last field shown.



The screenshot shows a form titled "CREATE / MODIFY DATABASE DESIGN". It contains several input fields and labels: "Database: ANYBASE" (text box), "Fieldname:" (text box), "Field type: P" (text box), "Indexed:" (checkbox), "Comment:" (text box), "Layout retained:" (checkbox), "Part-field:" (checkbox), "Subfields" (text box), "to" (text box), and "Valid values (pattern: , in database.field:)" (text box).

Figure 2–27 The Field Definition form

The keys you will need to fill this form are shown below.

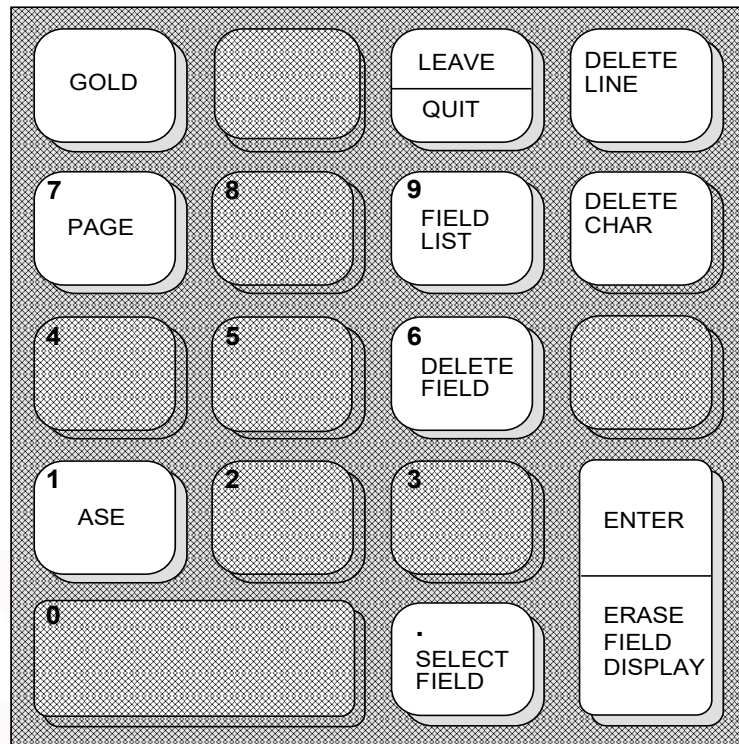


Figure 2-28 The Field Definition form design keys

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, and may include the underscore (_).

Move to 'Field Type'. You must enter at least the first two characters of the TRIP data type you wish to assign to this field (**TE**xt, **PH**rase, **NU**mer, **IN**teger, **DA**te, **TI**me or **ST**ring).

For example, the **PH**rase field **Anyfield** would be entered like this:

```

CREATE / MODIFY DATABASE DESIGN

Database: ANYBASE

Fieldname: anyfield   Field type: ph
Indexed:              Comment:
Layout retained:
Part-field:
Subfields            to
Valid values (pattern: , in database.field: )
  
```

Figure 2-29 Naming and typing a field

Press <Tab>, <→ Arrow>, <↓ Arrow> or ↓ to obtain the rest of the screen.

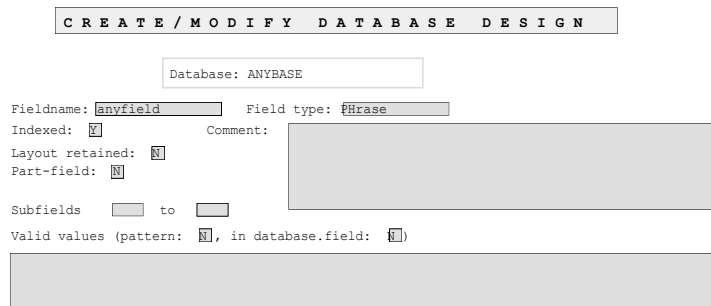


Figure 2–30 The Field Definition form for a PHrase field

If you have used shortened designations for field type, TRIP automatically expands them into their full type names when the screen is repainted. TRIP also presents only the field attribute boxes which are appropriate to the field type you have chosen, as illustrated by Figures 2–21 through 2–27.

The system displays the default values for each of the single character entry fields, Y (Yes) and N (No). The mode is always overwrite in a one-letter entry field. The defaults are Yes for 'Indexed' (with the exception of **STring**, which cannot be indexed), Yes for 'Layout Retained' if the field is a **TExt** field, and No for the remaining fields.

Sample Field Definition screens for the remaining six data types follow.

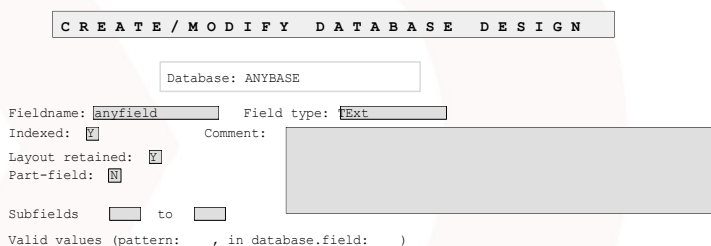


Figure 2–31 The Field Definition form for a TExt field

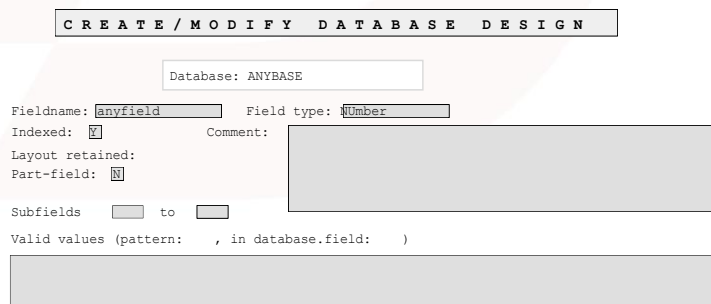


Figure 2–32 The Field Definition form for a NUmber field

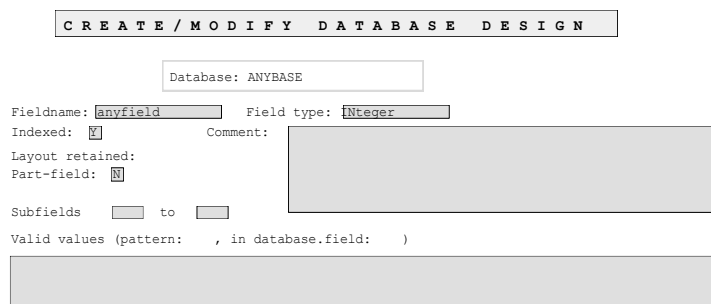


Figure 2–33 The Field Definition form for an INteger field

CREATE / MODIFY DATABASE DESIGN

Database: ANYBASE

Fieldname: anyfield Field type: Date

Indexed: ☒ Comment:

Layout retained:

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 2–34 The Field Definition form for a DAtе field

CREATE / MODIFY DATABASE DESIGN

Database: ANYBASE

Fieldname: anyfield Field type: Time

Indexed: ☒ Comment:

Layout retained:

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 2–35 The Field Definition form for a Tlme field

CREATE / MODIFY DATABASE DESIGN

Database: ANYBASE

Fieldname: anyfield Field type: SString

Indexed: Comment:

Layout retained:

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 2–36 The Field Definition form for a SString field

Field Attributes and Restrictions

The following table presents the Field Definition form’s attributes and restrictions (and Yes/No defaults if defined) for each of the field types **TExt**, **PHrase**, **NUmber**, **INteger**, **DAtе**, **Tlme** and **SString**.

Attribute	TE	P H	NU	IN	D A	TI	ST
Field name	✓	✓	✓	✓	✓	✓	✓
Field type	✓	✓	✓	✓	✓	✓	✓
Indexed	Y	Y	Y	Y	Y	N	Ø
Comment	✓	✓	✓	✓	✓	✓	✓
Layout retained	Y	N	Ø	Ø	Ø	Ø	Ø
Part field	N	N	N	N	N	N	N
Subfields	✓	✓	✓	✓	✓	✓	✓
Valid value type	Ø	✓	Ø	Ø	Ø	Ø	Ø
Valid value	Ø	✓	✓	✓	✓	✓	Ø

Table 2–15 Field attributes and restrictions

Y=Yes

N=No

✓=Applicable for this data type

Ø=Not applicable for this data type

Field Attributes

These include the Indexed, Comment, Layout Retained, Part Field and Subfield definitions.

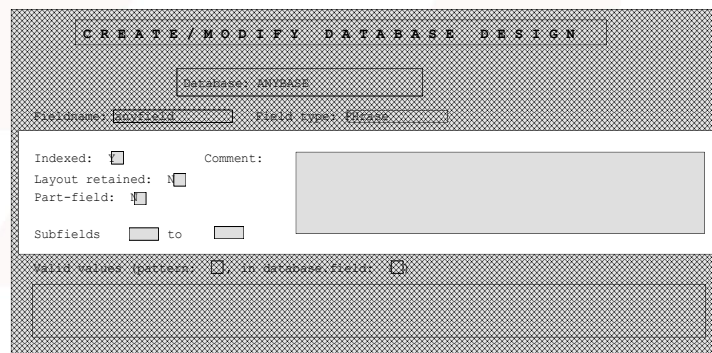


Figure 2–37 The Field Attributes

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.

This is an auto-tab field, in which progression to the next field is automatic after entering one of the choices listed below. The indexing categories are Y (Yes), N (No), W (Word Indexing) and S (Separate Indexing).

Yes (Y)

This 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.

STring 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.

No [Indexing] (N)

Fields in this category are not indexed, and thus not searchable. This could save file space for those applications that contain data unlikely to ever be searched.

Word Indexing (W)

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.

Separate Indexing (S)

This 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 2-29). 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.

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:

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

Restrictions

These include the Valid Values fields.

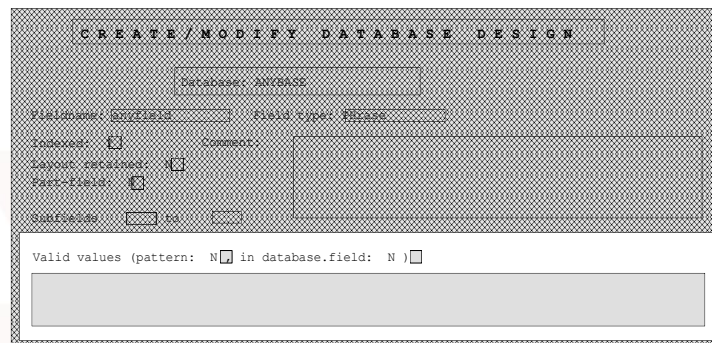


Figure 2–39The Field Attributes

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:

Valid values

You may specify a *list of accepted values* for a field by entering the elements of the list in the entry field at the bottom of the screen, separated by commas. Each element represents the contents of a subfield. The list may contain at most twenty elements, for a maximum length of 255 characters.

Note:

TRIPsystem expects that any values will be listed immediately below the Valid Values line on the TRIPclassic Field Attributes form. If this first line is left blank, then any definitions on subsequent lines will not be saved.

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.

A pattern is defined by answering Yes in the *Pattern* box and entering the valid pattern in the entry field at the bottom of the screen. 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,0,9	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 2–16 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 2–17 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 2–18 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 2–19 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 2–20 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 2–21 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.

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.

The default for DAtE fields is that the year minimally must be specified. This can be overridden by setting one of the following values in the list of valid values:

YMD	Year, month and day all must be specified
YM	Year and month must be specified
Y	Only year needs to be specified (this is the default)

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.

Dictionaries

Entering 'Yes' in the design field labelled 'in database.field' signifies that the field currently being defined is to be restricted by a data dictionary, which is held in a specific field of a particular database.

For example, entering

Alice.person

in the Valid Values design field means that entries are validated against the field **person** in the database **Alice**. To specify that entries must *not* occur in the dictionary, use the **NOT** modifier, e.g.

NOT Alice.person

in the Valid Value field.

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.

Note:

The contents of a data dictionary can be viewed during data entry by pressing <Gold><D> in the entry box associated with this field.

Overlay Forms For Field Definition

Field Definition overlays include Field List, ASE and Accounting forms.

The Field List

After defining or modifying one or more fields, you can list all fields created to date using the **<Field>** key (**<kp 9>**).

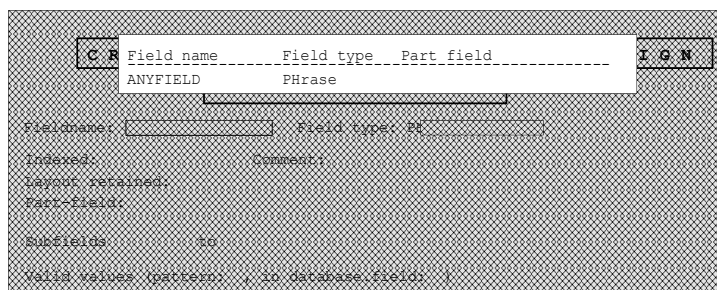


Figure 2-40 The Field List overlay for Field Definition

To select a field from the list for closer inspection or alteration, use the **<↑ Arrow>** and **<↓ Arrow>** cursor keys (and **<Next Page>** or **<kp 7>** if the list is long), and press **<Select>**, or **<kp .>**. To save any changes, press **<Enter>**. To leave the list without selecting, press **<Leave>**.

To alter a field definition, you may also type the field name in the Field Name design field and press **<Return>**. This will retrieve and display all relevant design information for that field, in the same manner as selecting it from the field list.

ASE

Use **<kp 1>** to display the ASE overlay form.

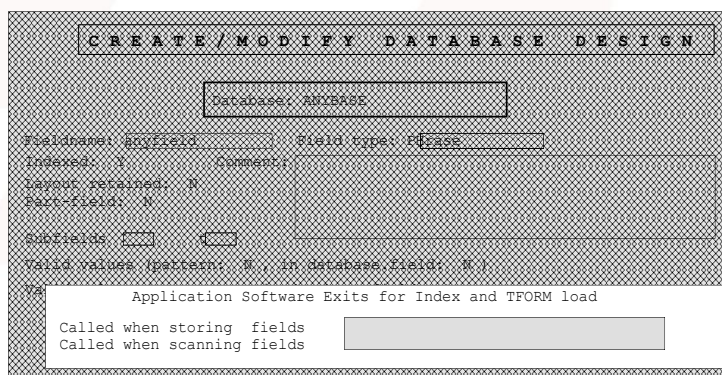


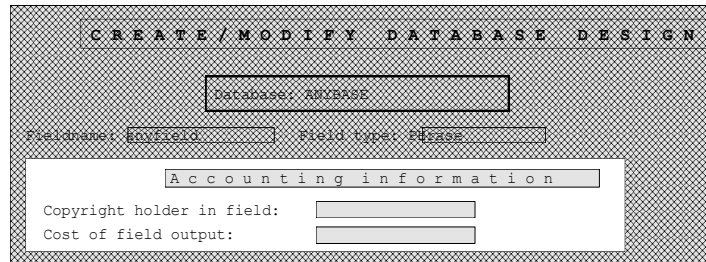
Figure 2-41 The ASE overlay for Field Definition

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

Press **<Enter>** to save, or **<Leave>** to exit without saving.

Accounting Information

This overlay allows the definition of a record copyright holder and a unit cost for examining the onscreen and printed contents of that field. Press **<Page>** (**<kp 7>**) to access this form.



The screenshot shows a terminal window titled "CREATE / MODIFY DATABASE DESIGN". Inside, there is a box labeled "Database: ANVBASE". Below it, "Fieldname: anyfield" and "Field type: P4base" are displayed. A sub-header "Accounting information" is centered. Under this header, there are two labels: "Copyright holder in field:" and "Cost of field output:", each followed by an empty input field.

Figure 2–42 The Accounting Information overlay

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.

Press **<Enter>** to save, or **<Leave>** to exit without saving.

Saving the Database Design

Saving a new database design entails entering it as an item in the **Control** system file that controls the database. Save an entire design specification by pressing **<Enter>** 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 the 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 2–22 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 in Windows or \$TDBS_EXE/bafini in UNIX 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.

Follow the pathway for database creation from the Primary Option Menu:

Primary Option Menu:
Administration ↵
 Databases ↵
 DB Design ↵
 Create/Modify Database ↵
 Create/Modify Database Design
Form↵

Choose the *Create/Modify Database* option.

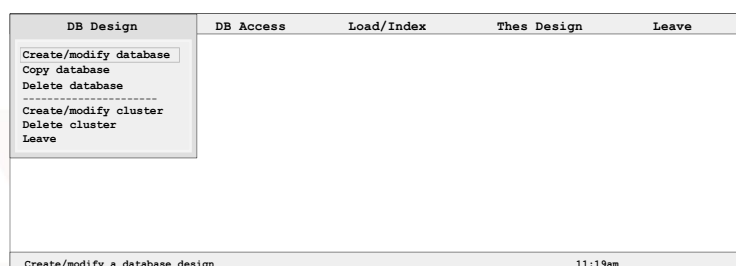


Figure 2–43 The Create/Modify Database option

Fill in the name of the database you wish to alter and press ↵.

CREATE / MODIFY DATABASE DESIGN

Database:

Figure 2–44 The Create/Modify Database Design form

To remove a field from the database design, bring up the Field Definition form and choose the field to delete from the field list, or type the name of the field in the 'Field Name' field; then press <Delete Field> or <kp 6>.

Note:

If you have previously opened the database that you are now modifying, you will have to exit TRIP and reenter before the changes that you make become apparent in all instances. Do this by using the CCL command

BASE xyz

This is because, for performance reasons, TRIP caches the design of any database that you open in any given session, and does not reread them until you start a new session.

Deleting a Design

Follow the pathway for database deletion from the Primary Option Menu:

Primary Option Menu:

Administration ↵

Databases ↵

DB Design ↵

Delete Database ↵

Delete Database Design Form ↵

Choose the *Delete Database* option.

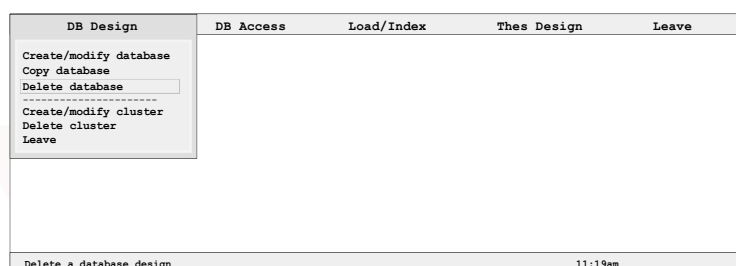


Figure 2–45 The Delete Database option

Fill in the name of the database you wish to delete and press ↵. Press <Enter> to confirm.

DELETE DATABASE DESIGN

Database:

Figure 2–46 The Delete Database Design form

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 you attempt to delete a database which has content, TRIP will issue an error message and abort the deletion. To delete such a database, you must first delete the database's **BAF**, **BIF** and **VIF**, after which all associated forms and formats will also be deleted.


Copying a Design

You can copy a design using the *Copy Database* menu.

Follow this pathway from the Primary Option Menu:

Primary Option Menu:
Administration ↵
 Databases ↵
 DB Design ↵
 Copy Database ↵
 Copy Database Design Form ↵

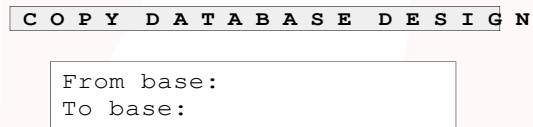
Choose the *Copy Database* option.



The screenshot shows a menu titled 'DB Design' with several options. The 'Copy database' option is highlighted. The menu also includes 'Create/modify database', 'Delete database', 'Create/modify cluster', 'Delete cluster', and 'Leave'. The status bar at the bottom indicates 'Copy a database design' and the time '11:19am'.

Figure 2–47 The Copy Database option

Fill in the name of the database you wish to copy and the name of the database to create, and press ↵.



The screenshot shows the 'COPY DATABASE DESIGN' form. It has two input fields: 'From base:' and 'To base:'. The status bar at the bottom indicates 'Copy a database design' and the time '11:19am'.

Figure 2–48 The Copy Database Design form

You can then alter the new database design using screen forms. To create the files **BAF**, **BIF**, **VIF** and **Log** of the copy database, you must use the alternative *Create/Modify Database*, and save the design by pressing **<Enter>** from the *General Database Properties* form. 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.

Note:

*Output format and data entry forms are not copied from the old database to the new when using this option. These must be copied separately from their own administration drop-down menu, or by **EXPORTing** and **IMPORTing** via CCL.*

Related 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 ↵

```
(st)
Data base: ALICE                Design created: 1992-10-19    18:04:02
Owner:      SYSTEM              Design revised: 1993-05-23   22:41:47

Number of records: 475          Last update:    1987-06-18    8:43:44
                                   Last index:     1993-05-24    8:50:29

Description: Text from 'Alice in Wonderland' and 'Through the Looking-Glass'
            by Lewis Carroll

Default format:      FULL
Formats available:   1, 2, SHORT, FULL

Files: TRIP$DEMO:ALICE.BAF
       TRIP$DEMO:ALICE.BIF
       TRIP$DEMO:ALICE.VIF

Submit queue:      TOBSS$BATCH                                ..>>
```

(Enter your command, please.) (Database) ALICE

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit 11:39am

Figure 2–49 Status for database Alice, screen one of two

```
(st)
Submit queue:      TOBSS$BATCH
Notify on completion: Y
Print log file:    N
Keep log file:     Y
```

Field name	No	Type	Part	Mand	Indx	Orig	Cost	Comment
CHAPTNR	1	INTEGER	N	Y	Y	N	0	Chapter number
CHAPTER	2	PHRASE	N	Y	Y	N	0	Chapter heading
PERSON	3	PHRASE	N	N	Y	N	0	Persons acting in the text
SPEAKER	4	PHRASE	N	N	Y	N	0	Persons speaking in the text
TXT	5	TEXT	N	N	Y	N	0	Text of the chapter
VERSE	6	TEXT	N	N	N	N	0	Text of the verse
TXT2	7	TEXT	N	N	Y	N	0	Text appearing after the verse

(Enter your command, please.) (Database) Alice

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit 11:39am

Figure 2–50 Status for database Alice, screen two of two

Here the **BAF**, **BIF** and **VIF** name specifications are given, as well as the names of the output formats 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 **Status** 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 FILE=filename ↵
```

For example:

```
EXPOrt BASe=mybase FILE=myfile.def ↵
```

or use

```
EXPOrt BASe=mybase.* FILE=myfile.def ↵
```

to export the database design and any associated output formats and entry forms for **Mybase** as well.

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 FILE=filename ↵
```

For example:

```
IMPOrt BASe=mybase FILE=myfile.def ↵
```

or use

```
IMPOrt BASe=mybase.* FILE=myfile.def ↵
```

to import the database design and any associated output formats and entry forms for **Mybase** as well.

Database Clusters

Creating a Cluster

You can define static or permanent database clusters (as opposed to dynamically from CCL) from the menu option *Create/Modify Clusters* using this pathway:

Primary Option Menu:
Administration ↵
 Databases ↵
 DB Design ↵
 Create/Modify Cluster ↵
 Create/Modify Database Cluster
Form ↵

Choose the *Create/Modify Cluster* option from the *DB Design* menu.

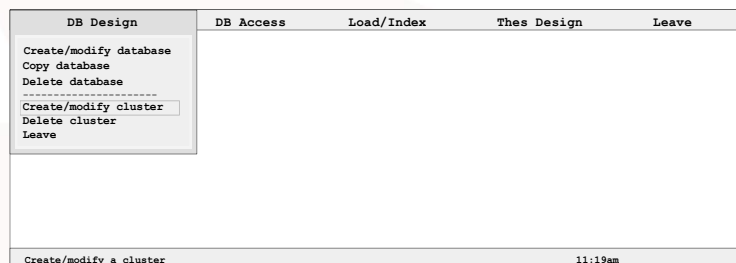


Figure 2–51 The Create/Modify Cluster option

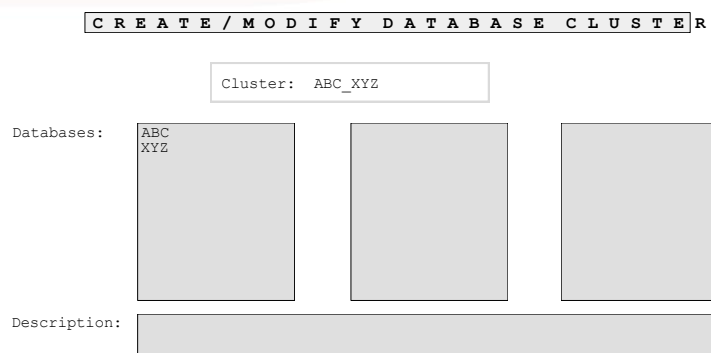
To create a cluster, type in a cluster or database group name and press ↵.



The screenshot shows a form titled 'CREATE / MODIFY DATABASE CLUSTER'. It has a single text input field labeled 'Cluster:'.

Figure 2–52 The Create/Modify Database Cluster form

TRIP will present you with a tabular form in which to list up to thirty database members.



The screenshot shows the 'CREATE / MODIFY DATABASE CLUSTER' form. The 'Cluster:' field contains 'ABC_XYZ'. Below it, there is a section labeled 'Databases:' with three columns. The first column contains 'ABC' and 'XYZ'. Below the 'Databases:' section, there is a 'Description:' field.

Figure 2–53 The Cluster Databases form

There is also space provided for a description of the cluster, which will appear in CCL orders such as

Show BASE ↵

or

STatus clustername ↵

Exit and save to create the cluster. This can now be opened and searched like any other database.

Modifying a Cluster

Follow the pathway for cluster creation from the Primary Option Menu:

Primary Option Menu:

Administration ↵

Databases ↵

DB Design ↵

Create/Modify Cluster ↵

Create/Modify Database Cluster

Form↵

Choose the Create/Modify Cluster option.

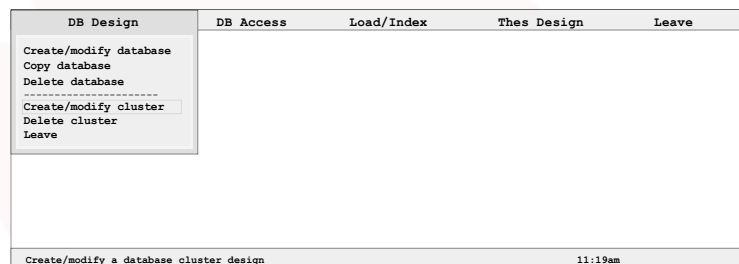


Figure 2–54 The Create/Modify Cluster option

Fill in the name of the cluster you wish to alter and press ↵.



The screenshot shows a form titled 'CREATE / MODIFY DATABASE CLUSTER'. It has a text input field labeled 'Cluster:'.

Figure 2–55 The Create/Modify Cluster Design form

Deleting a Cluster

Follow the pathway for cluster deletion from the Primary Option Menu:

Primary Option Menu:

Administration ↵

Databases ↵

DB Design ↵

Delete Cluster ↵

Delete Database Cluster Form↵

Choose the *Delete Cluster* option.

DB Design	DB Access	Load/Index	Thes Design	Leave
Create/modify database Copy database Delete database ----- Create/modify cluster Delete cluster Leave				
Deleting a database cluster design 11:19am				

Figure 2–56 The Delete Cluster option

Fill in the name of the cluster you wish to delete and press ↵. Press <Enter> to confirm.

D E L E T E D A T A B A S E C L U S T E R
Cluster: <input type="text"/>

Figure 2–57 The Delete Database Cluster form

Deleting a cluster has no effect upon its component databases.

Related CCL Commands

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

```
(st abc_xyz)
Cluster:  abc_xyz          Design created: 30-Jun-93   16:40:43
Owner:    SYSTEM          Design revised: 30-Jun-93  16:40:52
Number of records: 574
Databases in cluster: ABC, XYZ
Description: Cluster database of ABC and XYZ
Submit queue:      TDBS$BATCH
Notify on completion: Y
Print log file:    N
Keep log file:     Y
-----
(Enter your command, please.) (Database)
ABC_XYZ
Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit 11:39am
```

Figure 2–58 A S*Tatus* 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:

1. 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'.
2. 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'.
3. 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.
4. 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'.

5. 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 3–59 The 'Train' thesaurus, vertical representation

and here horizontally:

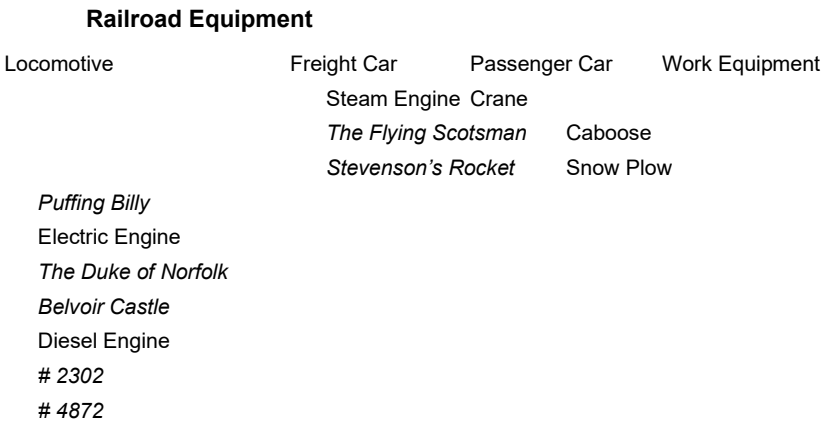


Figure 3–60 The 'Train' thesaurus, horizontal representation

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	<i>The Flying Scotsman,</i> <i>Stevenson's Rocket, Puffing</i> <i>Billy</i>
Electric Engine	Locomotive	<i>The Duke of Norfolk, Belvoir</i> <i>Castle</i>
Diesel Engine	Locomotive	# 2302, # 4872
<i>The Flying Scotsman</i>	Steam Engine	-----
<i>Stevenson's Rocket</i>	Steam Engine	-----
<i>Puffing Billy</i>	Steam Engine	-----
<i>The Duke of Norfolk</i>	Electric Engine	-----
<i>Belvoir Castle</i>	Electric Engine	-----
# 2302	Diesel Engine	-----
# 4872	Diesel Engine	-----
Crane	Work Equipment	-----
Caboose	Work Equipment	-----
Snow Plow	Work Equipment	-----

Table 3–23 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 an output format, entry form etc.

To create a thesaurus, follow this pathway to the ‘Create Thesaurus’ screen:

Primary Option Menu:

Administration ↵

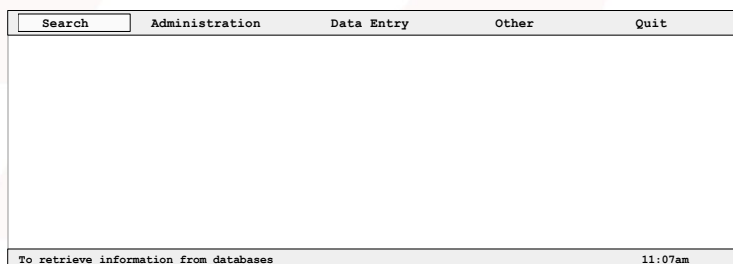
Databases ↵

Thes Design ↵

Create/modify ↵

Create/Modify Thesaurus Design Form

Beginning at the Primary Option Menu,



Search	Administration	Data Entry	Other	Quit
To retrieve information from databases 11:07am				

Figure 3–61 The Primary Option Menu

the cursor will be positioned in the upper left corner of the menu, with **Search** bolded or highlighted.

From **Search**, move the cursor right one cell using the <→ **Arrow**> key, and press ↵ or the <↓ **Arrow**> key. The *Administration* drop-down menu appears, with *Databases* the default cursor position.

Search	Administration	Data Entry	Other	Quit
	<div> Databases Users/Groups Forms/Procs Leave </div>			
Database/thesaurus maintenance 11:07am				

Figure 3–62 The Administration menu

If you wish to return to the Primary Option Menu at this point, press **<Leave>**, or **<Gold><Leave>** to quit TRIP altogether.

Press **↓** to choose the *Databases* option, which leads to a new menu headed by the cursor default *DB Design*. Cursor right four positions to *Thes Design*.

DB Design	DB Access	Load/Index	Thes Design	Leave
Designing thesauri 11:19am				

Figure 3–63 The Thesaurus Design option

Press **↓** or **<↓ Arrow>** for the *Thes Design* option list. Again, pressing **<Leave>** will return you to the Primary Option Menu; **<Gold><Leave>** exits TRIP.

DB Design	DB Access	Load/Index	Thes Design	Leave
			<div> Create/modify Copy Delete Leave </div>	
Create/modify a thesaurus design 11:19am				

Figure 3–64 The Create/modify option

Select the cursor default alternative *Create/modify* with **↓** to display the form for creating a thesaurus.

The first form you will need to fill asks for the name of the thesaurus to be built.

CREATE / MODIFY THESAURUS DESIGN
Thesaurus:

Figure 3–65 The Create/Modify Thesaurus Design form

If you have just created or modified a thesaurus via the *Thes Design* menu, the thesaurus name will be filled in on the access form. If not, you will need to provide one. Send your request by pressing **↓**.

Thesaurus Design

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 3–24 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	<i>except CT's parent or child</i> . This can be any other relative, or even a term with no ancestor in common with the CT
UF	other <i>non-CT</i> 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 3–25 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 3–26 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:

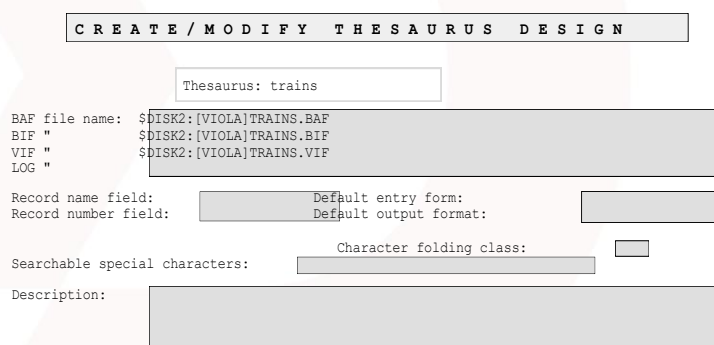
TRIP will not automatically detect a thesaurus design error wherein term 1A has NT=1B and term 1B has NT=1A. The same is true if term 1A has NT=1. Attempting to Display Down from a level above 1A can then result in a loop and possibly a crash if the Display MAXimum definition is sufficiently large. Care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

Database Design

General Thesaurus Properties

Having selected the option Create/Modify and entered the name you want to give the thesaurus, you will see the first page of the thesaurus design screen. The design forms are similar to the standard database design screens, with several slight differences.

Refer to Chapter Two of this guide, 'Creating a Database', for background information on designing a thesaurus.



CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

BAF file name: \$DISK2:[VIOLA]TRAINS.BAF
BIF " \$DISK2:[VIOLA]TRAINS.BIF
VIF " \$DISK2:[VIOLA]TRAINS.VIF
LOG "

Record name field: Default entry form:
Record number field: Default output format:

Searchable special characters: Character folding class:

Description:

Figure 3–66 The General Thesaurus Properties form

Physical Files

During the physical design of the thesaurus, you may wish to designate a device or directory other than the default or current one for the storage of the **BAF**, **BIF** and **VIF**.

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 output format is the same as for database design.

Be cautious in designing customized output formats for use with a search form, as the same output format 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 **Status** requests as with a standard database.

Overlay Forms, General Thesaurus Properties

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

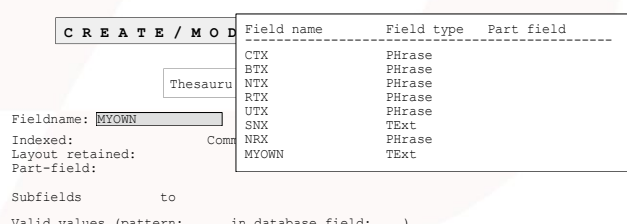
Field Definition

To create fields in addition to those provided in the thesaurus template or modify the attributes of the pre-existing fields, press **<Next Page>** to display the 'Field Definition' form. Any customized 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 here.

Pressing **<Field>** (**<kp 9>**) with the cursor somewhere in the 'Field Definition' form will display the current field list for the thesaurus.



Field name	Field type	Part field
CTX	PHrase	
BTX	PHrase	
NTX	PHrase	
RTX	PHrase	
UTX	PHrase	
SNX	TExt	
NRX	PHrase	
MYOWN	TExt	

Figure 3–67 Naming and typing extra thesaurus fields

Pressing **<Select>** while the cursor is resting beside one of these fields imports the property information for that field into the field definition form for review or modification.

For example, selecting **CTX** and pressing **↓** imports the **CTX** field definition information, and shows that it must consist of exactly one subfield (Subfields 1 to 1 on the screen), as seen below:

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: CTX Field type: Phrase

Indexed: ☒ Comment: Controlled term

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 3–68 The field definition form for CTX

The field definition screens for **BTX**, **NTX**, **RTX**, **UFX** and **SNX** follow.

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: BTX Field type: Phrase

Indexed: ☐ Comment: Broader term

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 3–69 The field definition form for BTX

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: NTX Field type: Phrase

Indexed: ☐ Comment: Narrower term

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 3–70 The field definition form for NTX

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: RTX Field type: Phrase

Indexed: ☐ Comment: Related term

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: , in database.field:)

Figure 3–71 The field definition form for RTX

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: UFX Field type: Phrase

Indexed: ☒ Comment: Used for: synonyms

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: N, in database.field: I)

Figure 3–72 The field definition form for UFX

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: SNX Field type: Text

Indexed: ☒ Comment: Scope note: Term description

Layout retained: ☒

Part-field: ☒

Subfields to

Valid values (pattern: N, in database.field: I)

Figure 3–73 The field definition form for SNX

CREATE / MODIFY THESAURUS DESIGN

Thesaurus: trains

Fieldname: NRX Field type: Phrase

Indexed: ☒ Comment: Hierarchical term number

Layout retained: ☒

Part-field: ☒

Subfields to 1

Valid values (pattern: N, in database.field: I)

Figure 3–74 The field definition form for NRX

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 **Status** 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 this 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.

Status

Use **STatus** to review general information about your thesaurus once the design has been completed and saved:

STatus databasename ↵

(st)

Thesaurus: THESALI

Owner: SYSTEM

Design created: 1993-05-24 9:45:45

Design revised: 1993-06-14 9:30:51

Number of records: 140

Last update: 1993-06-02 28:10:49

Last index: 1993-06-02 28:11:06

Description: A thesaurus of the persons in the demonstration data base ALICE

Formats available: LISTSV, LISTALL

Files: TRIP\$DEMO:THESALI.BAF

TRIP\$DEMO:THESALI.BIF

TRIP\$DEMO:THESALI.VIF

Submit queue: TDBS\$BATCH

Notify on completion: Y

Print log file: N

Keep log file: Y

(Enter your command, please.)

(Database)
THESALI

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit

11:39am

Figure 3–75 STatus for thesaurus ‘Thesali’, screen one of two

(st)

Submit queue: TDBS\$BATCH

Notify on completion: Y

Print log file: N

Keep log file: Y

Field name	No	Type	Part	Mand	Indx	Orig	Cost	Comment
CTX	1	PHRASE	N	Y	Y	N	0	Controlled term
BTX	2	PHRASE	N	N	N	N	0	Broader term
NTX	3	PHRASE	N	N	N	N	0	Narrower terms
RTX	4	PHRASE	N	N	N	N	0	Related terms
UFY	5	PHRASE	N	N	Y	N	0	Used for: synonyms
SNX	6	TEXT	N	N	Y	N	0	Scope note: Term description
NRX	7	PHRASE	N	N	Y	N	0	Hierarchical term number

(Enter your command, please.)

(Database)
Thesali

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit

11:39am

Figure 3–76 STatus for thesaurus ‘Thesali’, screen two of two

Show

Show BASe ↵

Show BASe LIST ↵

IMPOrt/EXPOrt

IMPOrt THESaurus=thesaurusname file=filename ↵

EXPOrt THESaurus=thesaurusname file=filename ↵

Chapter 4: System Logging Functions

System logging functions include accounting and auditing.

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).

Assigning Field Costs

Assigning a cost to a field when a record is output is done in the database specification. To access the cost accounting form, press <Page> (<kp 7>) from the database design field specification form.

See Chapter Two, 'Database Design' for more information regarding field cost assignment.

Activating System Logging 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 for UNIX, 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.

Logical Names

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

The first, TDBS_ACCDIR defines a directory in which the system accounting logs are kept.

TRIP assumes the accounting file to be kept in TDBS_SYS if the environment variable 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 tdb.conf.

The second logical name or environment variable, TDBS_ACCFLG can be used to customize the name and content of accounting logs.

Any definition of TDBS_ACCFLG in a user's environment will be overridden by a definition in tdb.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 TDBS_SIF.

For example, if in UNIX the environment variable TDBS_SIF is defined as /usr/users/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 or 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.

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 following *line types*:

Line Name	Line Acronym	Starting Position	Ending Position	Length ³	Field Description	When Recorded
Beginning	B-line ¹	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 ²	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 FR equency orders	Immediately after order has been completed
MeasuRe	R-line	52	253	200	Logs ME asure orders	Immediately after order has been completed
Search	S-line	52	253	200	Logs FI nd 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 4–27 Line types and the accounting log

¹ 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.

² The operating system statistics captured by the *E-Line* include total TRIP connect time and CPU time.

³ 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	<code>base corr</code>
	<code>F \$rip</code>
	<code>show</code>
	<code>base alco=alice,corr</code>
	<code>find alice</code>
	<code>show</code>
	<code>show reverse</code>
	<code>freq rname</code>
	<code>meas chaptnr</code>
	<code>f jabber\$ or jan</code>
Batch Print Job	<code>print file=x</code>

The accounting log file would appear as follows:

COLUMN POSITION	
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

Figure 4-77 Sample accounting file

Part 2:

Forms



Chapter 5: 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.

Notes:

- On UNIX, the default screen size for data entry forms is fixed at 24 rows (23 writeable) by 80 columns and is not affected by the default VT terminal screen size of 25 rows by 80 columns; therefore defining a new terminal screen of a size other than the default needs careful consideration. For more information on defining a non-standard TRIPclassic terminal, see information on *TERMINAL* (UNIX only in the Environment Configuration Setup document).
- On Windows, the TRIPclassic terminal screen size is fixed at the default 25 x 80 size and cannot be altered.

Screen Navigation

For keys with two functions (designated by a dividing line in the numeric keypad illustration that follows), the function below the line is activated by pressing <**Gold**> before pressing the key itself.

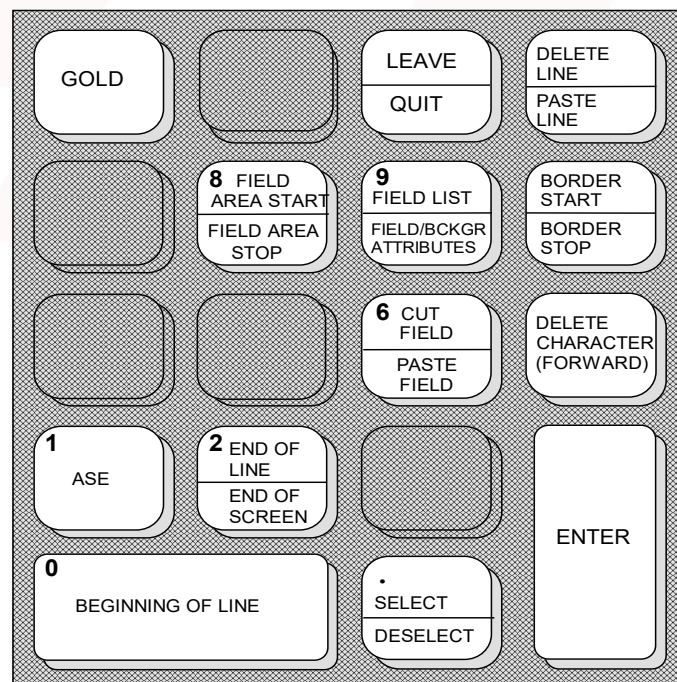


Figure 5-78 Data entry form design keypad functions

Entry Form Components

A form consists of *entry fields*, each entry field or box is assigned to a field in the database, and of *background texts*, i.e. headers for the entry fields and other texts for user information on the form. You can select visual attributes for the background texts as well as for the entry fields.

Creating a Data Entry Form

You can create several data entry forms for one database. One of them can be used as the default entry form, which must be named and defined as the default in the database design.

To create an entry form from the Primary Option Menu, follow this pathway:

Primary Option Menu:
Administration ↵
 Forms/Procs ↵
 Entry forms ↵
 Create/modify ↵
 Create/Modify Data Entry Form

To start creating an entry form you choose Administration in the top menu,

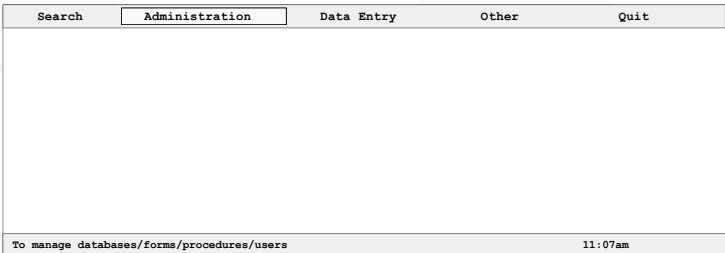


Figure 5–79 The Primary Option Menu

and below that the option *Forms/Procs*.

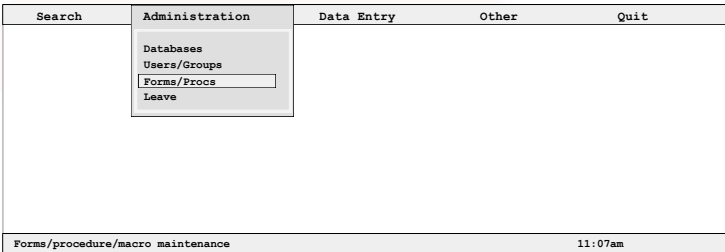


Figure 5–80 The Administration menu

Select Entry forms,

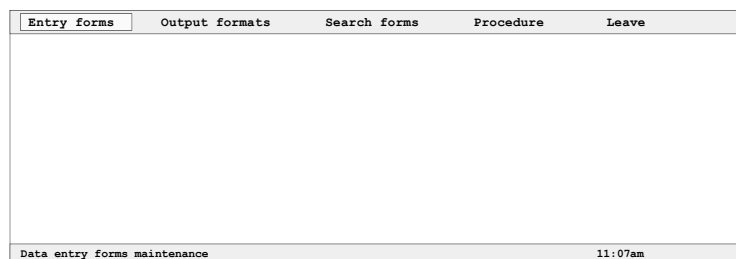


Figure 5–81 The Data Entry Forms Maintenance menu and its sub option *Create/modify*.

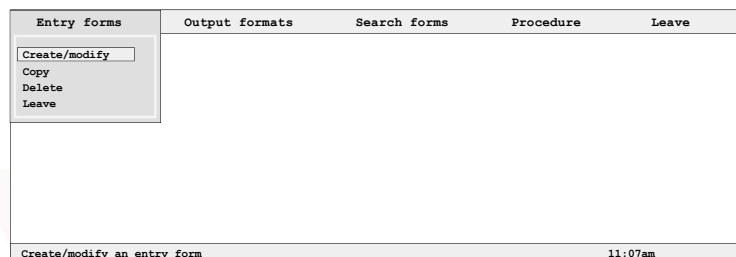


Figure 5–82 The Create/modify option

You will be asked for the name of the database and the name of the entry form.

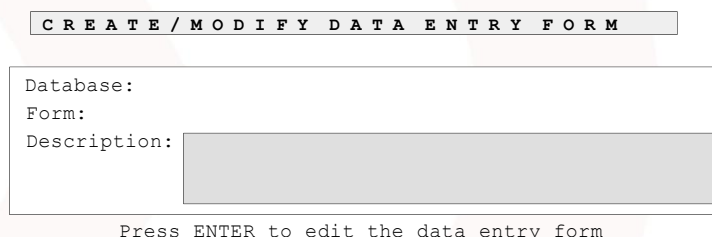


Figure 5–83 The Create/Modify Data Entry form

You can now also assign a description to a form which will appear with CCL orders like **Show EFORM**.

When you have entered the details and the system has accepted the database name, you will see a screen that is empty except for a status line at the bottom telling you that this is the *Data Entry Form Definition - Screen Layout Phase*.

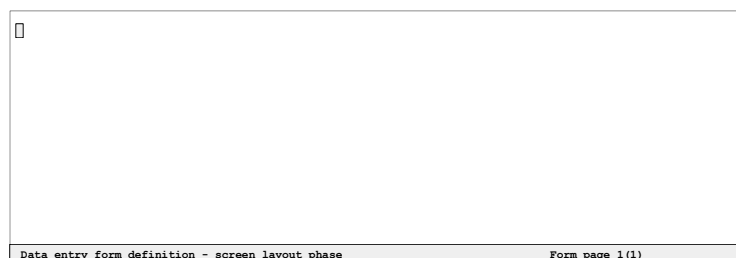


Figure 5–84 The Data Entry Form Definition screen

In your form you may use the first twenty-three lines and all eighty columns of the screen (the twenty-fourth line is used for status messages from the system). Each entry form may consist of several screen pages, and is

protected from editing or deletion by any users other than the form's creator or the file manager for that file.

The recommended order of entry form creation is as follows:

1. Create fixed screen text (entry box label)
2. Draw entry box
3. Select field
4. Insert constants (if any)
5. Assign field data options.

Each will be addressed separately in the following sections.

Fixed Screen Text

Background texts can be placed anywhere on the screen pages, outside the entry field areas. By default background text is shown with normal attributes and without a border, but you can alter the visual attributes in the following way.

Placing the cursor outside the entry field areas and pressing **<Gold><kp 9>** (**<Gold><Field List>**) will give you the list of attributes shown in the figure below.

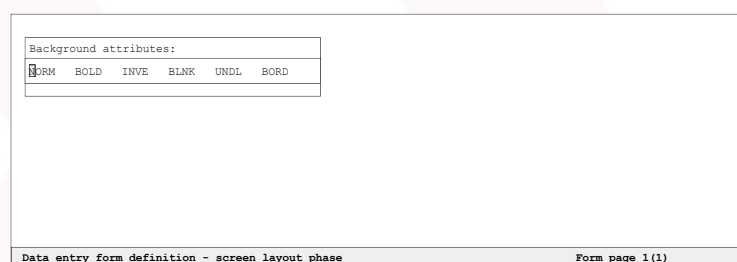


Figure 5–85 The Text Attributes menu

The visual background attributes include 'Norm' (Normal), 'Bold', 'Inve' (Inverse), 'Blnk' (Blink), 'Undl' (Underline) and 'Bord' (Border). Only the following combinations of visual background attributes are allowed:

- Bold/Underline
- Bold/Blink/Underline
- Inverse/Blink
- Inverse/Underline
- Blink/Underline.

Note:

Selecting border here surrounds the whole form with a border. For drawing a border around part of the form, see the section on 'Drawing Borders' later in this guide.

Use **<Tab>** or the cursor keys to move about the attribute list, and enter your choices by pressing **<Select>** (**<kp .>**). Press **<Gold><Select>** (**<Gold><kp .>**) to deselect, and **<Leave>** to return to the Screen Layout form with text attributes unchanged..

Background attributes:	
NORM	<u>BOLD</u> INVE BLANK <u>UDDL</u> BPRD

Data entry form definition - screen layout phase Form page 1 (1)

Figure 5–86 Text defined as ‘Bold, Underline’

Press <↓> to accept your selection and return to the screen layout.

The text you write now will have the visual attributes you selected, until you make a new attribute selection.

LAST NAME:

Data entry form definition - screen layout phase Form page 1 (1)

Figure 5–87 Bolded, underlined text

Defining a Field

Defining a Field Entry Area

A *data entry area* is any portion of the screen in which a user will be allowed to type, or enter data into.

To define the first entry area, place the cursor where you wish the [upper] left corner of the entry box to be and press <Field Area> (<kp 8>). Using the cursor keys and <Tab> to extend and compress the area; when it is the desired shape and size, press <Gold><Field Area> to end area definition.

For example, to create an entry area which is two rows by thirty columns, press the <↓ Arrow> once and the <→ Arrow> twenty nine times.

LAST NAME:

Data entry form definition - screen layout phase Form page 1 (1)

Figure 5–88 Entry box for ‘Last Name’

The maximum size of a data entry area is the size of the writable screen, twenty-three rows by eighty columns. If the background attribute is ‘Border’, the maximum screen size is reduced to twenty-one rows by seventy-eight columns.

Entry areas are easily scrolled in all directions during data entry, so their height need be only what is necessary to give the user a suitable view of the context. However, if an entry box for any data type except **Text** or **String** is

made too short, data that already exists can become truncated unless the 'Long' attribute has been set. For further information, refer to the section entitled 'Long Fields' appearing later in this guide.

Date fields must have a *minimum* of eight characters, four characters for year and two each for month and day, in the configuration YYYYMMDD (no separators are required). A complete date may have a *maximum* of eleven characters, four characters for year, three for month and two for day, or YYYYMMDD.

A **Date** may be entered in the short form 19930625 (YYYYMMDD - eight characters, no separators), but when a record is picked up for editing the data will be written in the full form selected for the installation or the individual user. If that form needs more characters than the entry field holds, the date will be truncated and may no longer be valid (see the 'Long' attribute presented later in this guide for more information).

The recommended format contains eleven characters, four characters for year, two each for month and day and two separators in the configuration YYYY/MM/DD.

Time fields have an eight-characters minimum, two characters each for hours, minutes and seconds plus two separators, or HH/MM/SS).

In the same way a **Time** value may be entered as 1140, without separators and seconds, but will appear in its full form if the record is picked up for editing.

If entry form screen space is limited, any field type except **Text** may be defined as 'Long' if subfields longer than the entry area are required. See the section on 'Long Attributes' for more information.

If you are defining a **Text** entry area which has 'Layout Retained' as a field attribute, make sure the entry box is the same width as the data contained in the **Text** field itself. For example, if the average sentence in this field has a Carriage Return <CR> every sixty-eight columns, the entry area must be at least *sixty-nine columns wide* to accommodate all text and formatting characters, or TRIP will not wrap the text correctly.

Selecting a Field

With the cursor resting in the field area to which you wish to link a field, press the <Field> key (<kp 9>) to view a list of the fields of the database.

Field name	Field Type	Part field	Used
LAST NAME	Phrase		
FIRST NAME	Phrase		
MIDDLE NAME	Phrase		
DATE OF BIRTH	Phrase		
YEARS OF SERVICE	Phrase		
CURRICULUM VITAE	Phrase		

Data entry form definition - screen layout phase Form page 1 (1)

Figure 5–89 The database field list

The list contains the names of the fields in the database design and their types. Use the <↑ Arrow>, <↓ Arrow>, <Tab> or <Return> keys to navigate the list via the 'Used' column. Press <Select> (<kp .>). The field selected will then be marked by an asterisk [*] in the 'Used' column,

Field name	Field Type	Part field	Used
LAST NAME	Phrase		*
FIRST NAME	Phrase		
MIDDLE NAME	Phrase		
DATE OF BIRTH	Phrase		
YEARS OF SERVICE	Phrase		
CURRICULUM VITAE	Phrase		

Data entry form definition - screen layout phase Form page 1(1)

Figure 5–90 Field selection made

and since you are allowed only one choice at a time in this list (each data entry area can belong to only one data entry field), you are immediately returned to the 'Screen Layout Phase' form.

To view the field selected, call up the field list as before. To edit your choice, place the cursor beside the asterisk of the selected field and press **<Gold><Select>** to undo the selection. The asterisk is removed and you can make a new selection.

Field Data Options

Screen Navigation

All of the data options described in this section use identical on-screen manoeuvring commands.

To specify the field properties of the entry area you are currently building, place the cursor inside the entry box and press **<Gold><Field>** (**<Gold><kp 9>**). Move through the attributes list vertically using **<Return>**, **<↑ Arrow>** or **<↓ Arrow>**, and horizontally with the **<Tab>** or **<← Arrow>** and **<→ Arrow>** keys. Enter your choices for each category by pressing **<Select>** (**<kp .>**), and deselect any option using **<Gold><Select>** (**<Gold><kp .>**).

LAS

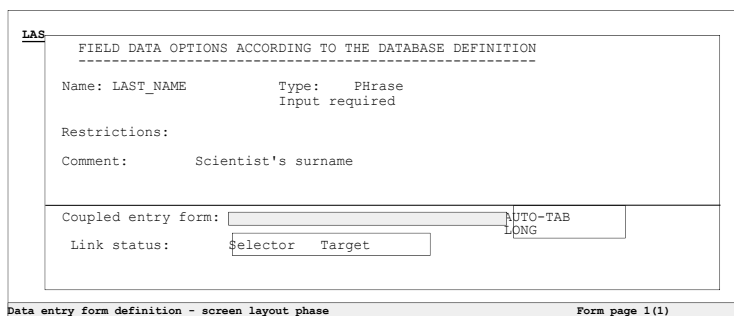
FIELD DATA OPTIONS		Field name: LAST NAME
-----		Field type: Phrase
Field attribute options:	NORM <u>HOLD</u> INVE BLNK UNDL BORD PROT	
Default value:	DATE TIME TSTAMP USER VMSUSER	
Default/target entry mode:	REPLACE INSERT APPEND	
Recall value of preceding record:	ALWAYS ON REQUEST	
Value checking:	Immediate	Flag first occurrence
Tuple number:	Other switches: <u>SHORT</u>	
Coupled entry form:	AUTO-TAB	
Link status:	LONG	
	Selector	Target

Data entry form definition - screen layout phase Form page 1(1)

Figure 5–91 The default Field Data Options form

Each option is discussed individually in the sections that follow. Save your selections by pressing **<Enter>**, after which you will be returned to 'Screen Layout Phase'. **<Leave>** closes the 'Field Data Options' form without saving option changes and returns you to 'Screen Layout'.

Pressing **<Next Page>** from 'Field Data Options' displays a list of the characteristics of the field you are working with.



LAS

FIELD DATA OPTIONS ACCORDING TO THE DATABASE DEFINITION

Name: LAST_NAME Type: PPhrase
Input required

Restrictions:

Comment: Scientist's surname

Coupled entry form: AUTO-TAB
LONG

Link status: selector Target

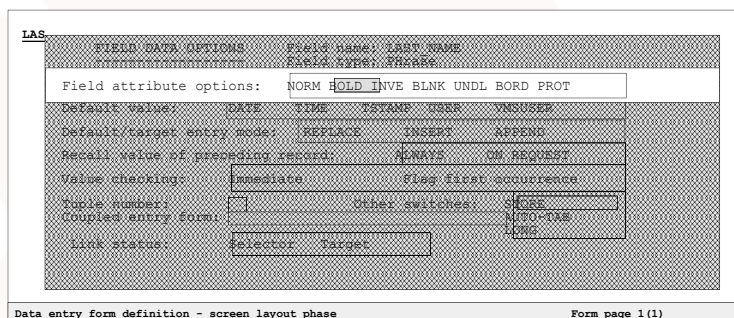
Data entry form definition - screen layout phase Form page 1(1)

Figure 5-92 The Field Characteristics summary

This information is derived from the database definition and cannot be changed from the entry form. Press **<Leave>** or **<Previous Page>** to return to the 'Field Data Options' form.

Field Presentation and Protection

'Field Attribute Options' includes both the six visual or descriptive attributes (field *appearance*) and a field's ability to be altered by the user (field *protection*). Both types may be modified.



LAS

FIELD DATA OPTIONS Field name: LAST_NAME
FIELD TYPE: PPhrase

Field attribute options: NORM ☒ HOLD ☒ INVE ☐ BLNK ☐ UNDL ☐ BORD ☐ PROT

Default value: DATE TIME TSTAMP USER VMSUSED

Default/target entry mode: REPLACE INSERT APPEND

Recall value of preceding record: ALWAYS ON REQUEST

Value checking: Immediate Flag first occurrence

Tuple number: Other switches: MORE

Coupled entry form: AUTO-TAB
LONG

Link status: selector Target

Data entry form definition - screen layout phase Form page 1(1)

Figure 5-93 The Field Attribute and Field Protect options

The visual field attributes include 'Norm' (Normal), 'Bold', 'Inve' (Inverse), 'Blnk' (Blink), 'Undl' (Underline) and 'Bord' (Border). Only the following field attribute combinations are allowed:

- Bold/Underline
- Bold/Blink/Underline
- Inverse/Blink
- Inverse/Underline
- Blink/Underline.

By default the entry area is bold unless specified otherwise in your terminal driver (see the Appendix in this guide for more information).

If you wish an area to have a border, remember that the border will occupy one extra character position in every direction.

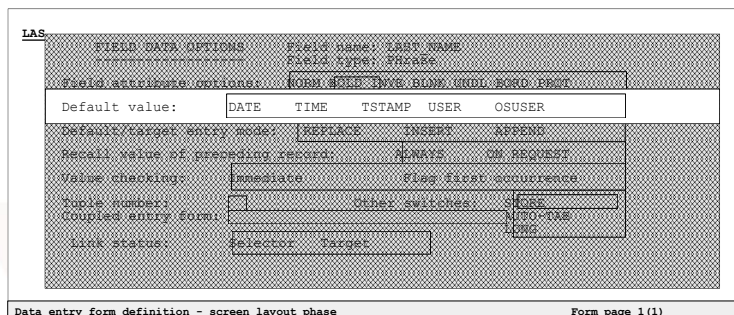
A protected field will prevent a user from entering data, and will be skipped as the cursor moves from field to field during data entry.

Defaults

Design-Time Default values

You may cause text to appear in an entry box as the default value by typing it in the appropriate entry area when creating (or modifying) the data entry form. It may be assigned any of these entry modes via the 'Field Data Options' form: REPLACE, INSERT or APPEND (see the section on 'Default Entry Mode' later in this chapter).

Run-Time Default values



The screenshot shows the 'FIELD DATA OPTIONS' form for a field named 'LAST NAME' of type 'PHRASE'. The 'Field attribute options' are set to 'NONE', 'FIELD TYPE BLOW UNDL', and 'BORD PROT'. The 'Default value' section has fields for DATE, TIME, TSTAMP, USER, and OSUSER. The 'Default/target entry mode' is set to REPLACE. Other options include 'Recall value of preceding record' (set to ALWAYS), 'Value checking' (set to immediate), 'Flag first occurrence' (checked), 'Table number' (set to 1), 'Coupled entry form' (set to NONE), and 'Link status' (set to Selector Target).

Figure 5–94 The Default values

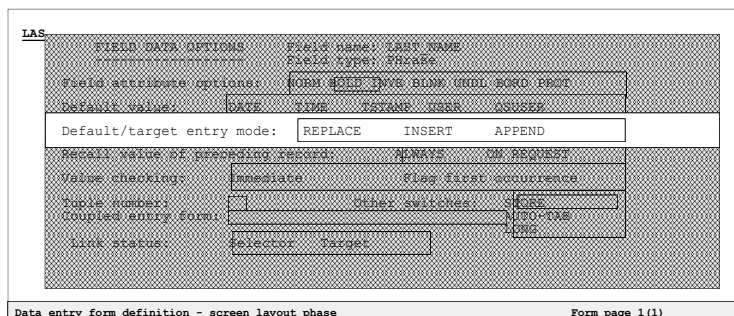
The following default values may be entered automatically by the system into fields of the entry form:

DATE	current date
TIME	current time
TSTAMP	current time-stamp (date and time)
USER	TRIP user name
OSUSER	operating system user name

Table 5–28 Default Entry Form Field Values

Default/Target Entry Mode

This option defines how default values requested previously (Date, Time, TStamp, User or OSuser) should be entered into each field. If no mode has been selected (the default setting), default values will be written to fields only upon record creation, not during modification.



This screenshot is identical to the previous one, but it highlights the 'Default/target entry mode' which is set to REPLACE. The other settings remain the same.

Figure 5–95 The Default/Target Entry mode

In the latter case they may be entered as the only subfield or paragraph of their fields, inserted at the beginning of the field or appended at the end of it, depending on the 'Entry Mode' chosen (see the following section for details).

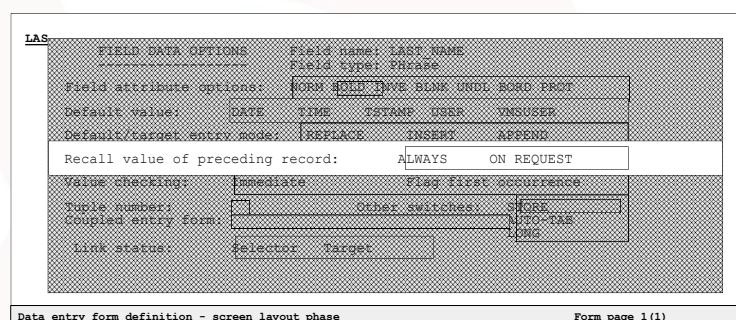
REPLACE [value]	as the only subfield (or paragraph) of the field
INSERT [value]	as a new first subfield (or paragraph) of the field
APPEND [value]	as a new subfield (or paragraph) at its end.

Table 5–29 Default Target Entry Modes

'Replace' deletes the entire contents of the subfield or paragraph, then inserts the new value, while 'Insert' positions the most recent data before all pre-existing information in the subfield or paragraph, and 'Append' places it after.

A target field in a linked database is 'Append' by default. Refer to the section entitled 'Link Status' later in this chapter for more information regarding target fields.

'Sticky' Fields



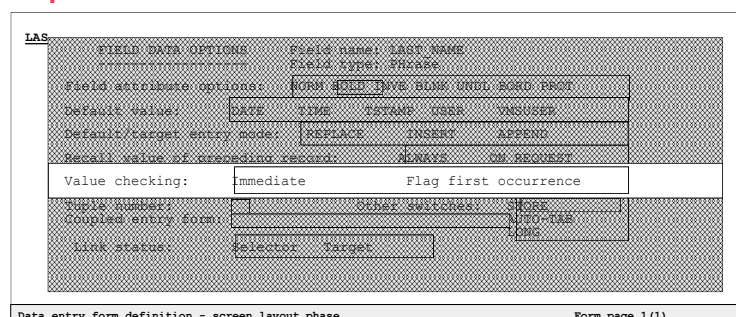
FIELD DATA OPTIONS Field name: LAST NAME
Field type: PHRASE
Field attribute options: NORM GOLD TIVE BLNK UNDL BORD PROT
Default value: DATE TIME TSTAMP USER VMSUSER
Default/target entry mode: REPLACE INSERT APPEND
Recall value of preceding record: ALWAYS ON REQUEST
Value checking: Immediate Flag first occurrence
Other switches: SHARE
Coupled entry form: AUTO-TIE
Link status: Selector Target

Data entry form definition - screen layout phase Form page 1(1)

Figure 5–96 The Recall Value of Preceding Record options

To save time when entering large numbers of similar records or making like modifications to many records, the contents of a field in the *preceding record* may be retained automatically (option 'Always'), or recalled when the user of the form orders it by pressing <Gold><R> (option 'On Request').

Validation Options



FIELD DATA OPTIONS Field name: LAST NAME
Field type: PHRASE
Field attribute options: NORM GOLD TIVE BLNK UNDL BORD PROT
Default value: DATE TIME TSTAMP USER VMSUSER
Default/target entry mode: REPLACE INSERT APPEND
Recall value of preceding record: ALWAYS ON REQUEST
Value checking: Immediate Flag first occurrence
Other switches: SHARE
Coupled entry form: AUTO-TIE
Link status: Selector Target

Data entry form definition - screen layout phase Form page 1(1)

Figure 5–97 The Value Checking options

Immediate Checking

Normally the values entered into a field are checked only when the record is completed and, when <Enter> is pressed, is sent to the BAF. To have the

contents of a field checked as soon as a user has completed entry into that field (signalled by pressing <Tab>), select the option 'Immediate'.

Flag First Occurrence

If an error is detected, you will get an error message and the cursor will be placed in the erroneous field. To help you detect spelling mistakes a field may also be specified 'Flag First Occurrence', in which the user will get a message from the system when a subfield content occurs for the first time in a **Phrase** field.

Note:

Values are checked against the database index.

Tuples

A field may be assigned to a data entry tuple, in which a series of fields are linked to form a table consisting of rows and columns. Each field is one column, and the collection of same-numbered subfields of those fields forms one row. A tuple may consist of any assortment of non-Text fields within the same entry form page.

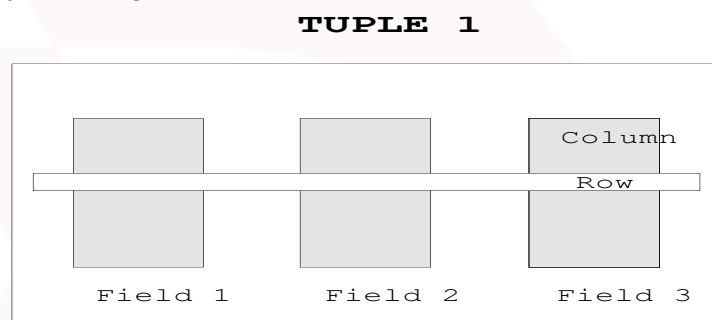


Figure 5–98 Tuple structure

Tuples may be useful in several situations. Tuples make entry of tabular data and empty subfields easier, and since each row is an individual entity, scrolling one field in a tuple causes parallel scrolling in every other field in the tuple.

For example, a record could contain a mailing list, with the fields **Name**, **Street**, and **City**. If the fields are assigned to a tuple, the task of entering items of the list is facilitated in several ways.

Navigating Tuples

Pressing <Tab> or <Backspace> from a subfield within the tuple moves the cursor to the subfield of the same number in the next or previous field of the tuple. When you enter the first item of the mailing list and press <Return>, a new line in the table is opened and the cursor is placed in the second subfield of the field **Name**. <Return>, pressed from any field of the tuple, always places the cursor in the next subfield of the first tuple field.

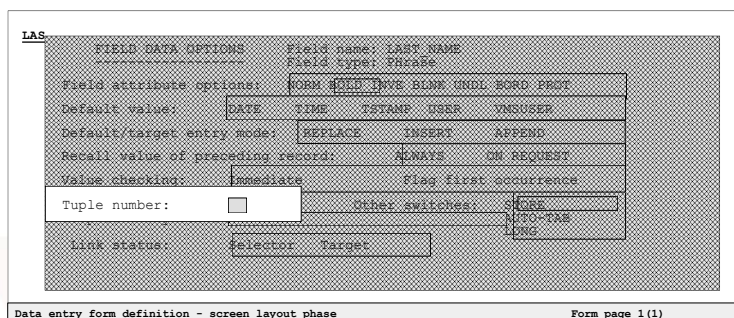
Within a tuple, the keypress <Delete Line> deletes the row the cursor is in. If you press <Undelete Line> (<Gold><Delete Line>) to restore the deleted portion, the deleted row will be inserted before the row where the cursor is positioned.

The <↑ Arrow> and <↓ Arrow> keys can be used to move about in the subfields of the same tuple subfield. With the cursor positioned at the

beginning and end of a tuple subfield, the **<← Arrow>** and **<→ Arrow>** keys move to the previous or next subfield in the same tuple. To leave a tuple, **<Tab>** to the last column of the table and press **<Tab>** again.

Creating a Tuple

To make a field a member of a tuple, enter the number of the tuple in the area labelled 'Tuple Number' on the Field Data Options form for this field. Each field that is to become a member of the tuple should be assigned the same number, which must be unique within the form.

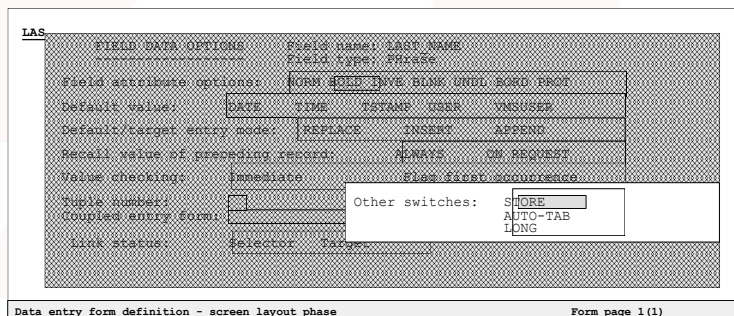


The screenshot shows the 'FIELD DATA OPTIONS' form for the field 'LAST NAME' (field type: PHRASE). The 'Tuple number' field is set to 1. Other options include 'Field attribute options' (NONE, FIELD, HAVE, BLNK, UNDL, BORD, PROT), 'Default value' (DATE, TIME, TSTAMP, USER, VMSUSER), 'Default/target entry mode' (REPLACE, INSERT, APPEND), 'Recall value of preceding record' (ALWAYS, ON REQUEST), 'Value checking' (Immediate, Flag first occurrence), 'Link status' (Selector, Target), and 'Other switches' (STORE, AUTO-TAB, LONG).

Figure 5–99 Tuple Number

Although several tuples may exist within one entry form, be careful not to intermingle different tuple numbers on the screen. Do not place a tuple between others of different numbers on the same screen; tupled fields must be in tab order (be adjacent fields) on the entry form. If they are not, you will need to either rearrange the entry areas so that the tupled fields adjoin, or change the field entry (tab) order.

Other Switches



The screenshot shows the 'FIELD DATA OPTIONS' form for the field 'LAST NAME' (field type: PHRASE). The 'Other switches' field is set to 'STORE'. Other options include 'Field attribute options' (NONE, FIELD, HAVE, BLNK, UNDL, BORD, PROT), 'Default value' (DATE, TIME, TSTAMP, USER, VMSUSER), 'Default/target entry mode' (REPLACE, INSERT, APPEND), 'Recall value of preceding record' (ALWAYS, ON REQUEST), 'Value checking' (Immediate, Flag first occurrence), 'Link status' (Selector, Target), and 'Tuple number' (1).

Figure 5–100 Other Switches

Data Shown But Not Stored

The default choice is 'Store', in which all data entered will be stored in the database. 'No Store' is useful for display-only fields. Use **<Select>** (**<kp .>**) and **<Gold><Select>** to remove the highlighting from 'Store' and change it to 'No Store'.

Auto Tab

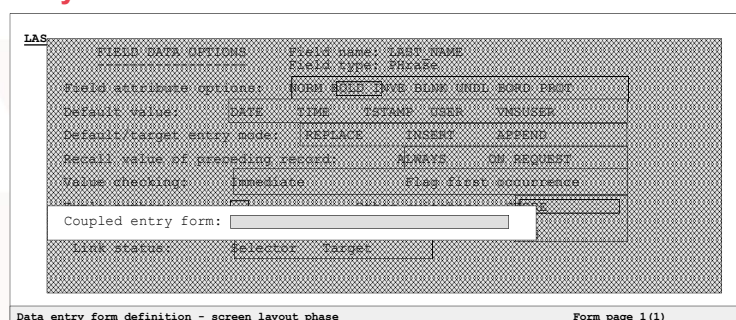
This feature is meant to be used when a field has a single subfield, where the entered data always contains the same number of characters. When 'Auto-Tab' is active, the cursor goes directly to the next data entry box as soon as the required number of characters has been entered.

The entry box must be the exact width of the field you wish to capture for 'Auto-Tab' to execute. If the entry box is ten columns wide and the field's value occupies only eight columns, you will need to manually <Tab> to the next field—automatic tabbing will occur only after the entry box has been filled to capacity. 'Auto-Tab' has no impact on **Text** fields even though it can be selected.

Long Fields

If a field is defined as 'Long', phrases of up to 255 characters may be entered. If a subfield is longer than the width of the corresponding data entry box, a diamond character [◇] indicates that there is more text available. When the cursor reaches the diamond, the next portion of hidden text slides into view.

Coupled Entry Form



IAS

FIELD DATA OPTIONS		Field name: LAST NAME	
Field type: PHRASE			
Field attribute options: NORM GOLD DATE BLANK UNDD BORD PROT			
Default value:	DATE	TIME	TSTAMP USER VMSUSER
Default/target entry mode:	REPLACE	INSERT	APPEND
Recall value of preceding record:	ALWAYS	ON REQUEST	
Value checking:	Immediate	Flag first occurrence	
Coupled entry form: <input type="text"/>			
Link status:	Selector	Target	

Data entry form definition - screen layout phase Form page 1 (1)

Figure 5–101 Coupled Entry Form

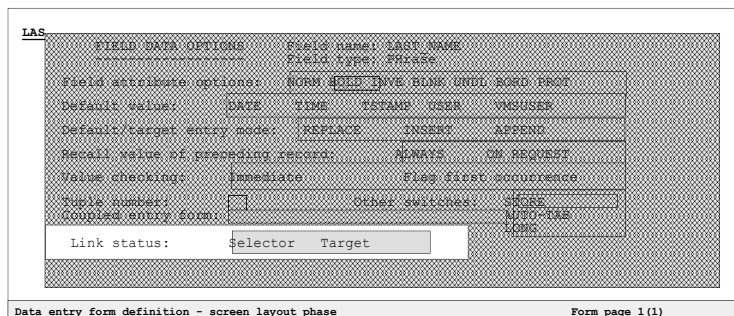
This option can be used when utilizing a non-restrictive data dictionary to validate field data entry. A user could call up the dictionary database entry form to enter a new term in the dictionary without having to exit the current entry form and losing already filled in data.

By entering a database and entry form name separated by a period in the 'Coupled Entry Form' box, e.g. DICTIONARY.FULL, the entry form 'Full' for the database **Dictionary** would be activated when <Gold><T> is pressed while in the field entry box.

This field option can be used when the field is, either, defined in the database design to be validated against a field of another database, or the 'Link' selector for importing data from another database. In both cases, the value entered on the current form will be carried over to the called entry form. If the field is a 'Link' selector, the 'Link' target information is retrieved on return to the calling entry form.

In order for a user to create a new entry in the coupled database, he or she must have write access to the coupled database.

Link Status



Field name: LAST NAME
Field type: PHRASE
Field attribute options: NORM FIELD HAVE LINK UNDL WORD PROT
Default value: DATE TIME TSTAMP USER VMSUSER
Default/target entry mode: REPLACE INSERT APPEND
Recall value of preceding record: ALWAYS ON REQUEST
Value checking: Immediate Flag first occurrence
Table number: Other switches: CHECK AUTO-TAB LONG
Link status: Selector Target

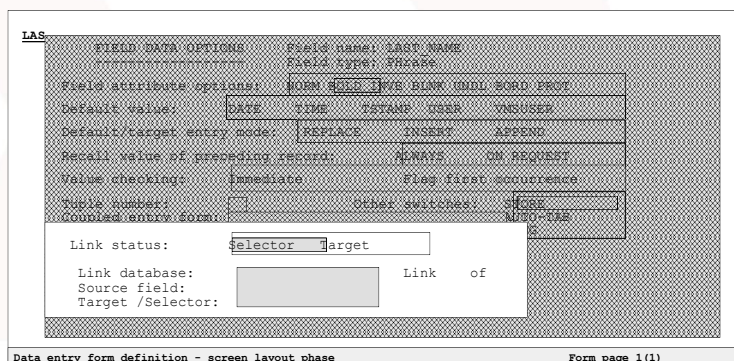
Data entry form definition - screen layout phase Form page 1(1)

Figure 5–102 Link Status

A *link* is a method of retrieving data from a secondary database, to be inserted into the record being entered into the current database.

Links are established by defining one field of a database as a *selector* (always a **PHrase** field), defining a *source database* from which to draw items related to the contents of that field and defining both a *source field* and a *target field* for each replacement.

To build a link, choose the option 'Selector' on the 'Link status' line in the field data options form of the field destined to be the selector. Press **<↓ Arrow>**, **↵** or **<Tab>** to commit your request.



Field name: LAST NAME
Field type: PHRASE
Field attribute options: NORM FIELD HAVE LINK UNDL WORD PROT
Default value: DATE TIME TSTAMP USER VMSUSER
Default/target entry mode: REPLACE INSERT APPEND
Recall value of preceding record: ALWAYS ON REQUEST
Value checking: Immediate Flag first occurrence
Table number: Other switches: CHECK AUTO-TAB LONG
Link status: Selector Target
Link database: Link of
Source field:
Target /Selector:

Data entry form definition - screen layout phase Form page 1(1)

Figure 5–103 The Link Definition form

The cursor moves to the 'Link Database' line, where you will enter the name of the intended source database. Press **<Return>**, at which point TRIP will validate that the source database contains a record name field. If it does not, you cannot define a 'Link' relationship with it.

The cursor then moves to the 'Source field' line. Press **<Field>** (**<kp 9>**) to view the list of the fields in the source database. Position the cursor beside the name of the field you wish to use and press **<Select>**. The contents of this field will be extracted for use in the current record at runtime.

You will be returned to the 'Link Definition Form'. Press **<Field>** once more from the 'Target/Selector' line to obtain a list of the fields with entry fields in the form you are working with. Select the name of your intended target field in this list. This is the field in which the extracted data from the source database will be written.

Press **<Enter>** to end the definition of the first link. The text "Link 1 of 1" will be written in the right corner of the form. As many additional link definitions

associated with the current selector field as desired can be established at this point.

To create another link in the same database, move the cursor to the source field and press **<Field>** (**<kp 9>**). Select a new source and target as before.

To create a link using the same selector in a different database, clear the 'Link database', 'Source field' and 'Target/Selector' areas by pressing **<Gold><Enter>**. With the cursor on the 'Link Database' field, choose new database, source and target fields and press **<Enter>**.

To browse in the list of links, press **<Gold><N>**, where the list will cycle from the first link established to the last, and from the last one to the first. To delete a link, press **<Cut Field>** or **<kp 6>**, and to empty the links form, press **<Gold><C>**. End the link definition by moving the cursor back to 'Selector' in 'Link Status' and pressing ↵ to save the entire Field Data Options form.

As in the case of default values, you may specify an entry mode for the target fields.

Note:

When creating links for validation purposes, TRIPsystem counts the selector and target as one field each; regardless of where the linked fields physically exist. Hence, if both target and selector are pointing to the same field in a link database, two links will be counted when reopening the entry form. This may seem obvious at first, but has been a cause of confusion on past occasions.

Form-Based ASEs

User-written subroutines or ASEs (Application Software Exits) may be linked to TRIP to check a record at the record level at the following points:

- before opening a record for data entry
- before writing a record to the **BAF**
- after a record has been written to **BAF**
- when leaving the record without writing to **BAF**

An ASE in data entry can also be called from defined function keys. If a function key is defined with the CCL order:

```
DEfine KEY keyname = \ASE routine name
```

e.g.

```
DEfine KEY F7=\myase
```

then the ASE can be activated during data entry by pressing that key.

Press **<kp 1>** from anyplace outside the entry area to display the record-level 'ASEs for Data Entry' form, which is used to establish the link to data entry when a routine is to act on the record level.

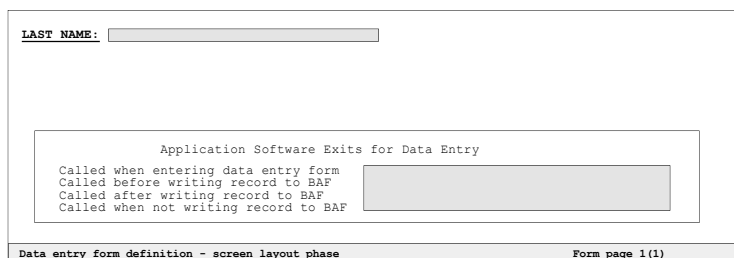


Figure 5-104 Record-level ASEs for Data Entry form

Enter the name(s) of the ASE(s) to call on the appropriate line(s) on the ASE form and press **<Enter>**. You may request only one ASE to be called per line.

Field-Specific ASE Overlay

User-written subroutines or ASEs (Application Software Exits) may also be linked to TRIP to check a record in data entry at these field-level points:

- before the cursor enters the field/subfield
- when the cursor leaves the field/subfield

Press **<Gold><9>**, then **<kp 1>** from anyplace inside the entry area to display the field-level 'ASEs for Data Entry' form, which is used to establish the link to data entry when a routine is to act on the field level.

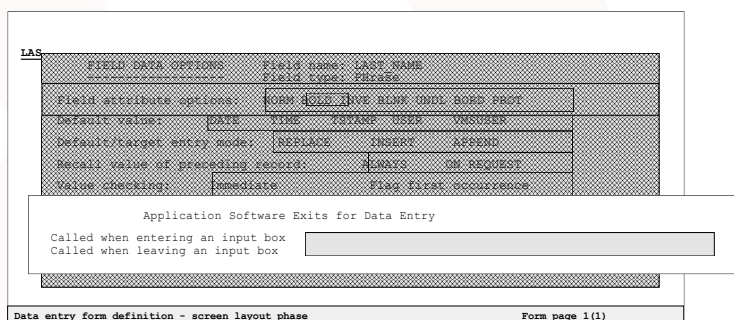


Figure 5-105 Field-level ASEs for Data Entry form

Moving Between Pages

To navigate through pages of the entry form, press **<Next Page>** to advance and **<Previous Page>** to reverse.

Cutting and Pasting

Use **<PF4>** and **<Gold><PF4>** to cut and paste text. Cutting and pasting does not affect entry area positions.

If you are not satisfied with the position of an entry field area after having pressed **<Gold><Field Area>** to end its definition, you place the cursor inside the area and press **<Cut Field>** (**<kp 6>**) to cut it. When it disappears from the screen the marking against its name in the list of fields also disappears.

To paste it in again you place the cursor where you want the upper left corner to be, and press **<Paste Field>** or **<Gold><kp 6>**. When you paste it in, the marking against its name in the list of fields will reappear.

Drawing Borders

Creating a Border

To surround a larger part of the screen page (a group of fields, perhaps, or a couple of lines containing instructions) with a border, place the cursor where you wish the upper left corner of the border. To begin and press **<Border>** (**<kp ->**). Use the **<→ Arrow>** key to draw the upper horizontal line, then the **<↓ Arrow>** to draw the perpendicular vertical line. When finished, press **<Gold><Border>** to save the new border.



Figure 5-106 Drawing the upper horizontal border

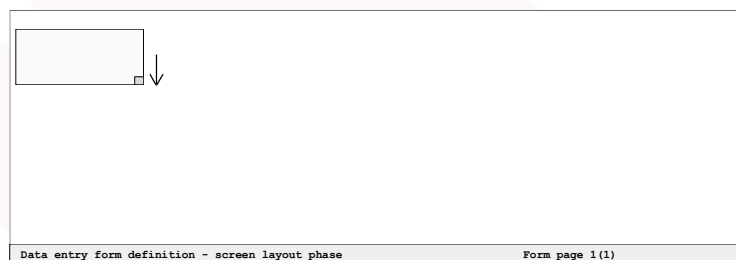


Figure 5-107 Drawing the perpendicular border

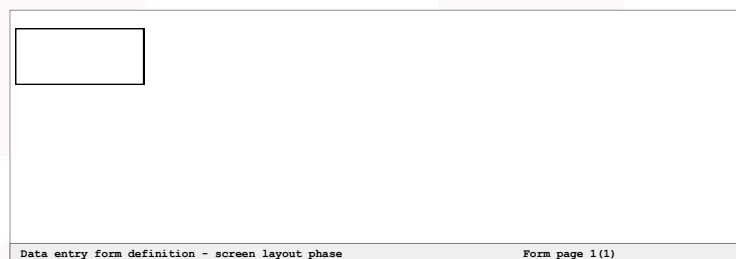


Figure 5-108 The finished border

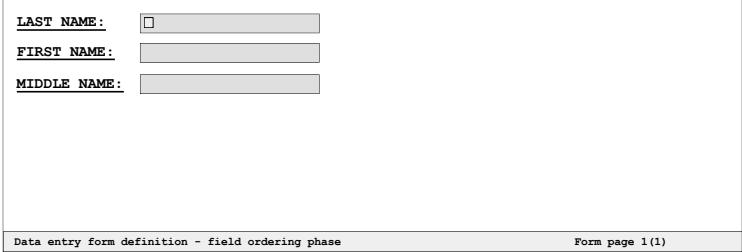
Deleting a Border

To delete a border, position the cursor on the upper left corner of the border to be removed, press **<Border>** and trace the border's perimeter *exactly* with using the cursor keys. Press **<Border>** once again to register the change.

Tab Order

By default, entry areas on a data entry form are traversed from left to right and from top to bottom on each form page.

To redefine the field order, press **<Gold><O>** (**<Gold><Order>**) while the cursor is positioned on the form page. The cursor will appear in the currently defined first field.



LAST NAME:

FIRST NAME:

MIDDLE NAME:

Data entry form definition - field ordering phase Form page 1(1)

Figure 5–109 The field ordering phase

Press **<Tab>** to move the cursor from field to field according to current field order.

When you press **<Select>** (**<kp .>**) the current field becomes the new first field. Move the cursor, using **<Tab>** or **<Backspace>**, to the field you want to be the second one, and press **<Select>** again. Choose the third field in the same way, and so on.

When you have ordered the fields to your satisfaction, press **<Enter>** to store the new order. If your field ordering did not include all the fields on the form page, the fields you left out will be appended after the fields you ordered.

To alter the new order, press **<Gold><R>** during the field ordering phase to restore the previous order. The field you select first after **<Gold><R>** will become the new first field, and so on.

Press **<Gold><C>** (**<Gold><Cancel>**) to restore the default order (left to right and top to bottom).

To save a new field order, press **<Enter>↓**. To exit Field Ordering and return to 'Screen Layout' without saving, press **<Leave>**.

All the fields of a tuple will be traversed before the cursor moves to a field outside the tuple, the normal order of movement being left to right and top to bottom. If you have assigned fields to data entry tuples, this pattern cannot be altered.

Saving the Form

Save the entry form by pressing **<Enter>** from one of its pages in the screen layout phase. To leave an entry form without saving it, press **<Leave>**, which will prompt you to save the entry form. Press **<Leave>** again to quit without saving.

Notes:

- *A user may have write access to selected fields only. In that case they will still be able to use an entry form containing the non-accessible fields, but not to write in those fields.*
- *To delete records you must have write-access to all fields of the database.*

Copying and Deleting Entry Forms

Choosing the option 'Copy' in the entry forms menu you can copy a data entry form, to use it either in an edited variety for the same database, or for another similar database.

You will be asked to identify the source form to copy (both the name of the form and the database to which it belongs), the name of the target database to which it will be copied and the name of the newly-copied form. If the form should be found unsuitable for the database you intend it for, a help text will let you choose between saving it regardless, editing it at once, or cancelling your copying order.

To delete an entry form you select the option 'Delete' in the entry forms menu and give the name of the form and the database it belongs to.

Related CCL Commands

Show

To list the entry forms that exist for a particular database, use:

```
Show EForm BAsE=databaseName
```

or just

```
Show EForm
```

Print

To have the list written to a printer or file, use the corresponding **Print** order:

```
Print EForm BAsE=databaseName
```

or just

```
Print EForm
```

These orders can be given by anyone who has write access to the database in question.

EXPORT/IMPORT

To **EXPORT** an entry form to a file, use:

```
EXPORT EForm=database.form FILE=filename
```

To **IMPORT** an entry form from a file, use:

```
IMPORT EForm=database.form FILE=filename
```

Chapter 6: Output Formats

An output format (also known as a report) 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 specialized 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 an output format or the database administrator (DBA) may edit or delete it.

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

The Format

An output format 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 format 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 output format before attempting to create the code, so that you have some idea of how it will (or should) look:

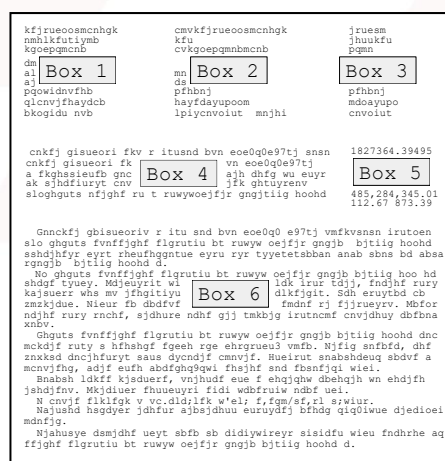


Figure 6–110 Format layout and construction

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

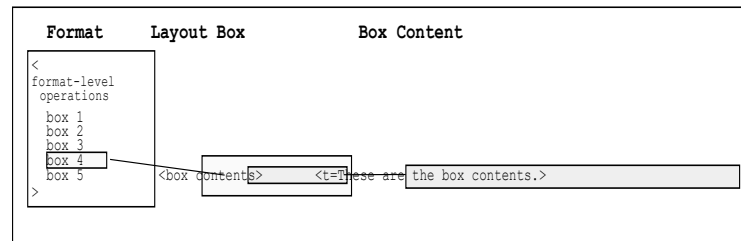


Figure 6–111 Output format components

The basic element of an output format 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

Format elements appear in a particular order within the output format:

1. The elements of the format 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.

Creating an Output Format

Before attempting to create a default output format for a database, you should include the default output format name in the database design.

To create the default output format, follow this pathway to the 'Create/Modify Output Format' screen:

Primary Option Menu:

Administration ↵

Forms/Procs ↵

Output Formats ↵

Create/Modify ↵

Create/Modify Output Format form ↵

Beginning on the Primary Options Menu, use the <➡ Arrow> once to highlight 'Administration':

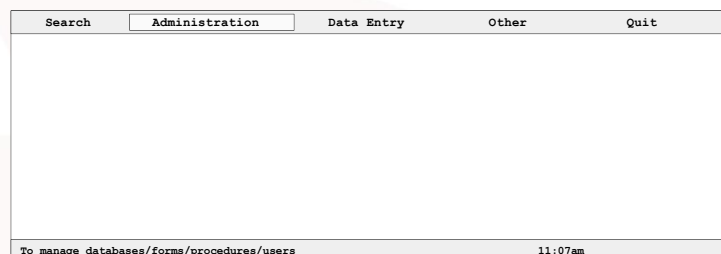


Figure 6-112 The Primary Option Menu

and press ↵ to pull down the 'Administration' submenu. Use <↓ Arrow> to highlight 'Forms/Procs':

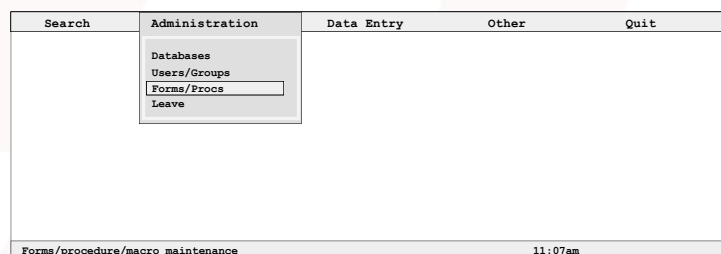


Figure 6-113 The Administration menu

and press ↵. When the Forms Maintenance Menu appears, use the <➡ Arrow> once to highlight 'Output Formats':

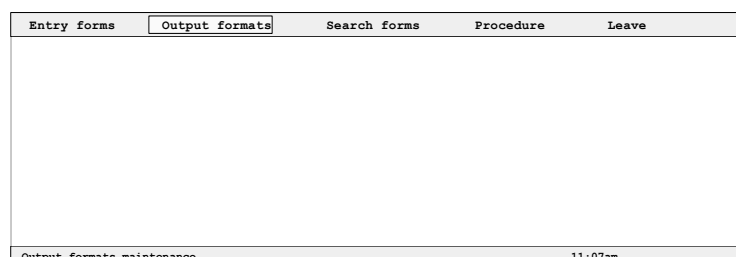


Figure 6-114 The Output Format option

and press <↵> to display the 'Output Formats' submenu. Press ↵ to activate 'Create/Modify'.

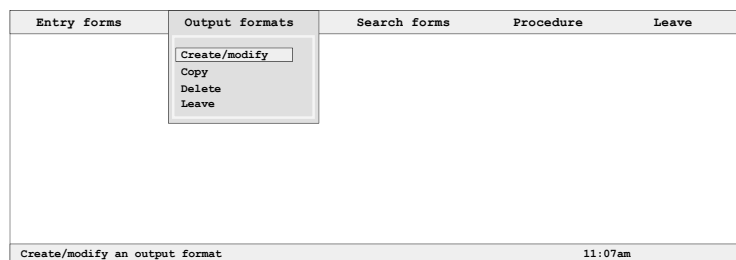
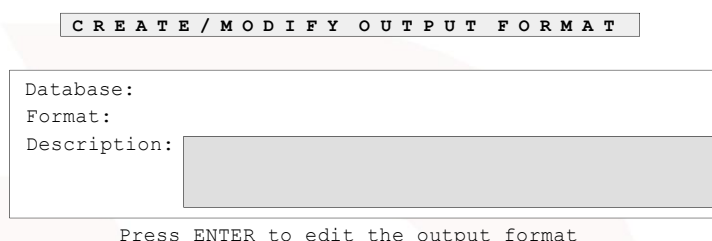


Figure 6–115 The Create/Modify option

When the 'Create/Modify Output Format' screen appears, enter the name of the database whose information the output format will display beside 'Database:' and the name of the output format to be created beside 'Format:'. You may also enter a description of the output format, which will appear in CCL orders such as Show FFormat. Press <Enter> to activate the form.



The screenshot shows a form titled 'CREATE / MODIFY OUTPUT FORMAT'. It contains three input fields: 'Database:', 'Format:', and 'Description:'. The 'Description:' field is highlighted with a grey background. Below the form, a prompt reads 'Press ENTER to edit the output format'.

Figure 6–116 The Create/Modify Output Format form

You are brought into the external (system) text editor used to create and modify the *specification file* for the output format. When this file is saved, TRIP checks the new or altered format, and unless it contains one or more syntax errors the format is then created. If TRIP does detect an error it will send a message detailing where and what it is, and you can edit the file.

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 output format 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 an output format 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 output format outlined above might look like this:

```
!      This is an output format:  
<  
<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 output format 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 an output format:  
<  
<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 output formats 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 output formats, 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:

Box at T(1),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:

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 output formats 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 output format 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 an output format, 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. <i>element</i>	labels or headlines that preface an element (field, subfield, paragraph or sentence)
Separator	s. <i>element</i>	separates the elements, and is not output before the first element or after the last one
Trailer	tr. <i>element</i>	outputs after each element has been printed

Table 6–30 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 6–31 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	<i>h.field or h.fieldtype or h.fieldname</i>
Subfield	<i>h.sub</i>
Paragraph	<i>h.p</i>
Sentence	<i>h.s</i>

Table 6–32 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	<i>s.field or s.fieldtype or s.fieldname</i>
Subfield	<i>s.sub</i>
Paragraph	<i>s.p</i>
Sentence	<i>s.s</i>

Table 6–33 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 output format:

**

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

A trailer, written as 'tr' in the format specification, follow the same rules and use the same elements as headers and separators:

To end a ...	Use ...
Field	tr.field <i>or</i> tr. <i>fieldtype</i> <i>or</i> tr. <i>fieldname</i>
Subfield	tr.sub
Paragraph	tr.p
Sentence	tr.s

Page 128 of 331

Field-Independent Text Strings

Headers, separators, and trailers are dependent on the field content; that is, if the entities they are meant to label are empty in one of the records, they will 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 6–35 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 6–36 Field type-dependent functions

Note:

These functions can be used only 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 an output format 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 (**T**ext fields first, then **P**hrase, 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

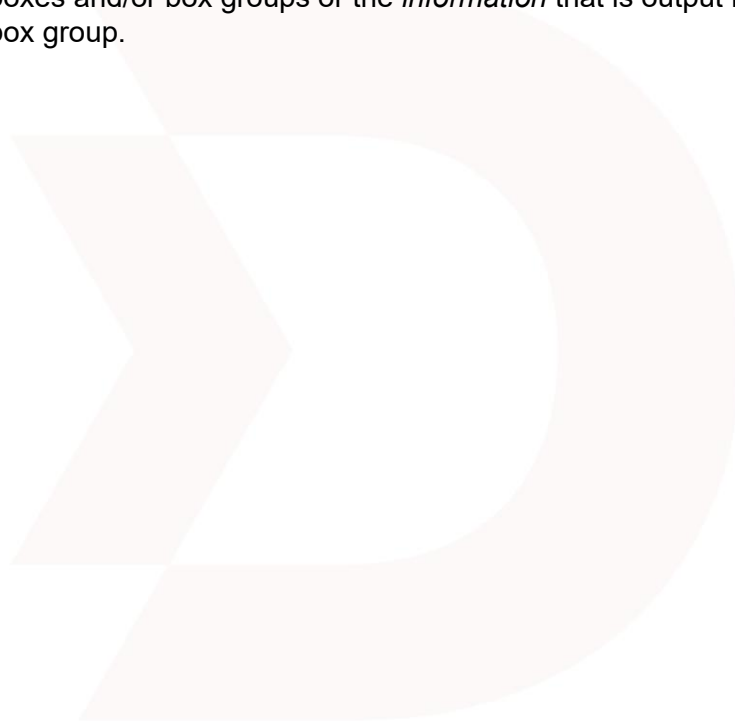
MODDATE (type DATE):
1: 17-Aug-93, 2: 17-Aug-93, 3: 18-Jun-93

```
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 <i>n</i> 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
<orig>	returns text field output to its original layout
<trace>	gives search history preceding the search result being output

Table 6–37 Box and box group functions

Format Functions

These functions have a global effect on the format, and all except **<noff>** 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
<sortfields>	defines fields to be used as sort keys

Table 6–38 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^{th} 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 6–39 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:

(st)

Data base: CARROLL
Owner: SYSTEM
Number of records: 24
Description: Text from 'Alice in Wonderland' and 'Through the Looking-Glass' by Lewis Carroll. This database is designed using part fields, each chapter is a record and within the record each paragraph is a part record.
Default format: 1
Formats available: 1, 2
Entry forms available: 1, FULL
Files: TRIP\$DEMO:CARROLL.BAF
TRIP\$DEMO:CARROLL.BIF

Design created: 1992-10-01
Design revised: 1993-05-24
Last update: 1990-02-06
Last index: 1993-05-24

20:50:05
8:31:13
15:16:29
8:53:32

(Enter your command, please.)

(Database)
CARROLL

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit

11:39am

Figure 6–117 SStatus Carroll, screen one of three.

(st)

TRIP\$DEMO:CARROLL.BIF
TRIP\$DEMO:CARROLL.VIF
Submit queue: TDBS\$BATCH
Notify on completion: Y
Print log file: N
Keep log file: Y

Field name	No	Type	Part	Mand	Indx	Orig	Cost	Comment
CHAPTER	2	PHRASE	N	Y	Y	N	0	
CHAPTNR	1	INTEGER	N	Y	Y	N	0	
PERSON	3	PHRASE	N	N	Y	N	0	
SPEAKER	4	PHRASE	Y	N	Y	N	0	
TXT	5	TEXT	Y	N	Y	N	0	
VERSE	6	TEXT	Y	N	Y	N	0	
TXT2	7	TEXT	Y	N	Y	N	0	

(Enter your command, please.)

(Database)
CARROLL

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit

11:39am

Figure 6–118 SStatus Carroll, screen two of three

(st)

Submit queue: TDBS\$BATCH
Notify on completion: Y
Print log file: N
Keep log file: Y

Field name	No	Type	Part	Mand	Indx	Orig	Cost	Comment
CHAPTER	2	PHRASE	N	Y	Y	N	0	
CHAPTNR	1	INTEGER	N	Y	Y	N	0	
PERSON	3	PHRASE	N	N	Y	N	0	
SPEAKER	4	PHRASE	Y	N	Y	N	0	
TXT	5	TEXT	Y	N	Y	N	0	
VERSE	6	TEXT	Y	N	Y	N	0	
TXT2	7	TEXT	Y	N	Y	N	0	
BOOK	8	PHRASE	N	N	Y	N	0	

(Enter your command, please.)

(Database)
CARROLL

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit

11:39am

Figure 6–119 SStatus Carroll, screen three of three

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

Page 138 of 331

```
<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 6–40 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.

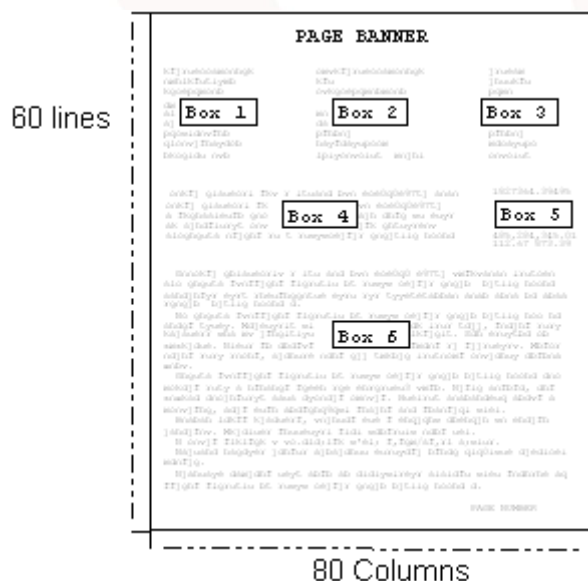


Figure 6-120 Paged output

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 output format 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.

Copying and Deleting Formats

It is often labour-saving to base new output formats on the structure of similar ones. If you need to adapt formats to a standard, you can do that easily using the option 'Copy' in the output format menu. You can use the copy option even if the databases are not alike enough to use exactly the same format - you can always edit the copy. When you save a copied format that is not fully

adapted to the database you copied it to, TRIP will ask you if you want to edit it at once, save it as it is, or cancel the copying order.

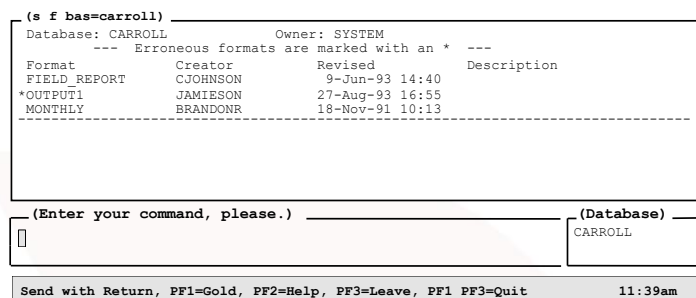
A format that is no longer used should be deleted, using the option Delete in the output format menu.

Related CCL Commands

To view a list of all the existing output formats for a database, give the order:

SHOW format BASE=*database name*

The **Show** window looks like this:



```
(s f bas=carroll)
Database: CARROLL      Owner: SYSTEM
--- Erroneous formats are marked with an * ---
Format      Creator      Revised      Description
FIELD_REPORT CJOHNSON      9-Jun-93 14:40
*OUTPUT1     JAMIESON      27-Aug-93 16:55
MONTHLY      BRANDONR      18-Nov-91 10:13
-----
(Enter your command, please.) (Database)
CARROLL

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit 11:39am
```

Figure 6-121

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/seperator/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, ..., [ selector.n ] : valuen
```

where the omission of the last selector (selector.n) signifies that when the field_element value does not match any given selector, the last value (value.n) 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 277 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 6–41 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
&	&
"	"
'	'
<	<
>	>

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-Text 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 xyzzY
```

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=TXT : >  
txt  
>  
>
```

and produces an output like this:

Record 1 content 'txt' in ALICE:

TXT : 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 the <noorig> function 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 TText 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 TText 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 6–42 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 which has 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:

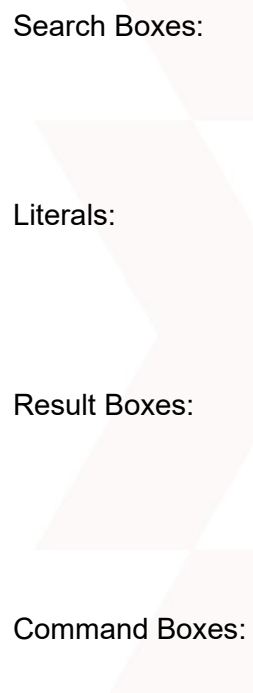
```
Record 36 has weight 97.
```

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.

Elements of a Search Form

A search form can assume nearly any configuration using these elements:

- 
- Search Boxes:** These are the boxes in which the user types his or her search expressions. Generally, each box on the search form is linked to one or more fields in one or more databases, and each may be scrolled horizontally so that combinations of expressions can be entered.
 - Literals:** These are the fixed texts written on the search form, such as the search form's title or box labels. It could, for example, indicate the type of search expression appropriate to each box, and it may also give help or hints as to the use of the search form as a whole.
 - Result Boxes:** These display the results of the searches. There is always a result line for the entire search form, which gives the outcome of all the search expressions of all boxes combined. There may also be a result box for each search box, which would give the findings of the search expression(s) for that particular box only.
 - Command Boxes:** These present all of the courses of action available at any point in time during a search form session. The command boxes on the main command menu might include such options as New Search, Modify Search, Show Result or Leave TRIP, each of which may have sub option boxes of their own. The menu boxes may change after choosing one or a succession of actions from the main menu.

Any number of the above elements can be combined to create search forms with any degree of complexity. A search form can be designed to call another search form via a command box option. For example, the search form **Alice_Demo** calls search form **Alice_Demo2** through its menu item 'Advanced Searching'.

The search form may also act as a menu system similar to the Primary Option Menu in the building of an application, and may contain such menu items as **Search**, **Data Entry** and **Choosing a Database**.

Anatomy of a Search Form

A search form consists of a Search Entry Window of nineteen lines by seventy-eight columns, a Form Report Line on line 20, a Command Menu Window on lines 21 and 22, a Form Message Line on line 23 and a TRIP Message Line on line 24. If a Search Report Window is included in the search form design, it is positioned either to the right or left of the Search Entry Window and is eight columns in width. The Search Entry Window then occupies only seventy columns.



The main parts or areas within a search form are shown below:

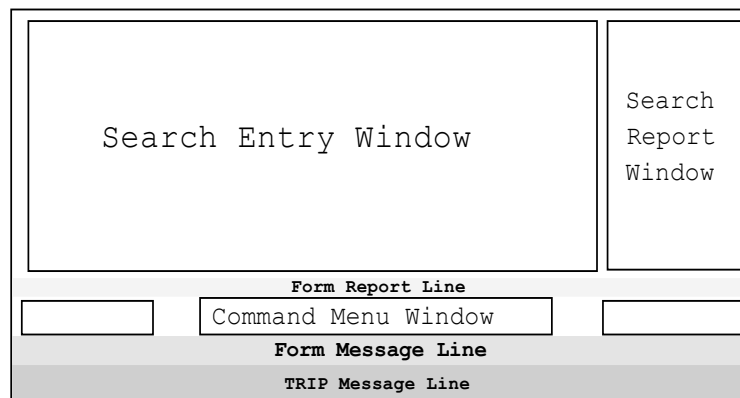


Figure 7–122 The anatomy of a search form

- The *Search Entry Window* can be seen as a data entry form, where search expressions are entered into search boxes. It occupies most of the screen area and holds both literals (title, box labels, instructions) and individual search boxes.
- To the right or the left of that is often an optional *Search Report Window*. This displays the number of hits for each search box in a corresponding report box, on the same line as the search box. After altering the contents of a search box and leaving it, a search is made for records or record parts that match the expression in the box. When the search has been completed, the number of records found by the search is displayed in the report box on the same line as the search box.
- The *Form Report Line* holds the total number of hit records found by the combination of all the entered search expressions. A logic clause in the form specifies how to combine the expressions, and this clause is evaluated when ↵ is pressed while the cursor is in the search window.
- Each screen can contain any number of boxes in its *Command Window* representing various courses of action, four to ten of which will be displayed at any one time.

Control is passed to the Command Window whenever a search is successful, or at any time by pressing <Gold><Tab> while the cursor is in the Search Window. One of the menu items then becomes the current command and is highlighted. However, another command may be selected with the help of the cursor keys.

The current command is executed by pressing ↵, and on its completion repeating ↵ moves the cursor in the manner prescribed for each menu item.

A typical cursor sequence might be:

1. Show Hits ↵
2. Scroll Forward—cursor remains at this option until all hits are shown
3. New Search ↵
4. search entry window is cleared.

- The *Form Message Line* informs the user of changes in search form status (default formats, position in output, etc.)
- The *TRIP Message Line* provides information on key assignments, form movement instructions, **Help** and so on.

A start procedure must be defined for every search form, which can perform various tasks such as open one or more databases, define views, scope, a thesaurus, etc. A **BASE** command in the form may also open databases. A search form is normally adapted to one or more databases, but is technically independent of them.

The search form **Alice_Demo2** is shown below as an example.

*** Advanced Searching in ALICE database ***

Chapter No.:

Chapter Heading:

Speaker in text:

Person in text:

Any other term(s) in text:

text=red

42

PLEASE ENTER SEARCH TERM(S) THEN PRESS <ENTER> OR PRESS <PF3> TO LEAVE

New Search

SHOW Menu

PRINT Options

Leave TRIP

Modify Search

Change Format

Goto CCL

Database Status

Enter Search Terms; Cursor Keys - Move; Enter - Search; PF2 - Help 11:35am

Figure 7–123 The search form Alice_Demo2

*** Advanced Searching in ALICE database ***

Chapter No.:

Chapter Heading:

Speaker in text:

Person in text:

Any other term(s) in text:

text=red

42

PLEASE ENTER SEAR

Short Format

Full Format

RESS <ENTER> OR PRESS <PF3> TO LEAVE

New Search

Change Format

PRINT Options

Leave TRIP

Modify Search

Goto CCL

Database Status

Enter Search Terms; Cursor Keys - Move; Enter - Search; PF2 - Help 11:35am

Figure 7–124 Alice_Demo2's 'Change Format' command menu

```

Record 169
Chapter: The Lobster Quadrille
Persons acting or referred to in the record:
    Mock Turtle,
    Gryphon
Chapter: The Lobster Quadrille
    Mock Turtle,
    Gryphon

The Mock Turtle sighed deeply, and drew the back of one flapper across his
eyes. He looked at Alice and tried to speak, but, for a minute or two, sobs
choked his voice. "Same as if he had a bone in his throat," said the
Gryphon; and it set to work shaking him and punching him in the back. At..>>

Total Number of Records Found by Complete Form 18
Show Results      Scroll Forwards      Next Record      MAIN Menu
Select Record     Scroll Backwards     Previous Record   Sort Output

Cursor Keys - Move; Return - Execute Option; PF2 - Help      11:35am

```

Figure 7–125 Alice_Demo2's 'SHOW Menu' command menu

```

*** Advanced Searching in ALICE database ***

Chapter No.: 
Chapter Heading: 
Speaker in text: 
Person in text: 
Any other term(s) in text: 
text=red 42

PLEASE ENTER SEARCH TERM(S) THEN PRESS <EN>
Print to file
Print and await completion
Print upon leaving TRIP

Total Number of Records Found by Complete Fo
New Search      SHOW Menu      PRINT Options      Leave TRIP
Modify Search   Change Format    Goto CCL           Database Status

Cursor Keys - Move; Return - Execute Option; PF2 - Help      11:35am

```

Figure 7–126 Alice_Demo2's 'PRINT Options' command menu

If search reports are used, each of the lines in the window may hold one search box that is from three to seventy columns wide, and within which up to 255 characters may be entered.

Areas of the search entry window that are outside the search boxes may be used for comments and instructions to guide the search form user.

Form searching is done with minimal use of function keys, but the following function key (sequences) are recognized:

Search Form Key(s)	Effect
<Gold><Tab>	toggles between the search entry window and the command menu window
<Next Page>	scrolls upward one page in Show , Display , Help , SStatus etc.
<Previous Page>	scrolls downward one page in Show , Display , Help , SStatus etc.
<Help>	accesses context-dependent help
<Leave>	exits the current context
<Select>	displays a <i>pick list</i> for item selection

Table 7–43 Search form movement keys

When used with the cursor in a search box, **<Select>** responds to a **Display** command in normal command mode searching, leading to a display of matching terms with the cursor positioned at the top.

The **<↑ Arrow>** and **<↓ Arrow>** keys move the cursor vertically within the display and cause it to scroll. Pressing **<Select>** adds the term beside the cursor to a list of picked terms. Pressing **↓** imports this list into the search box, where the terms are combined with a Boolean **OR**. However, as a search box holds a maximum of 255 characters, terms whose length cannot be physically accommodated are dropped.

<Select> can also be used to inspect and/or modify the contents of form variables, by choosing one or more of the alternatives displayed in response to a search form command. These can be used to change the current format, output destination, sort order and so on.

Most of the editing keys for data entry can be used to enter search expressions in the search window, for example cursor movement, deletion and undeletion, etc. However, the **<↑ Arrow>** and **<↓ Arrow>** keys are equivalent to **<Tab>** and **<Backspace>**, respectively.

Note:

If a TRIPclassic search form is invoked by a macro and if, in the process of searching, one attempts to move past the first / last record (such that the Top / Bottom of output messages are displayed), then upon leaving the search form the remainder of the macro will not execute.

Use of the CCL command **DEfine CONTinue** within the macro itself should enable the search form and the macro to work to completion.

Creating Search Forms

To create a search form, follow this pathway to the 'Create/Modify Search Form' screen:

Primary Option Menu:

Administration ↓

Forms/Procs ↓

Search Forms ↓

Create/Modify ↓

Create/Modify Search Form ↓

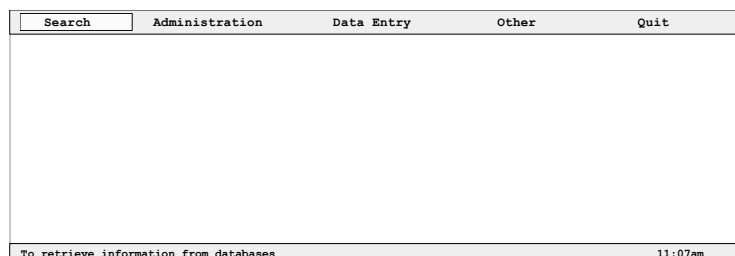


Figure 7–127 **The Primary Option Menu**

Search	Administration	Data Entry	Other	Quit
	Databases Users/Groups Forms/Procs Leave			
Forms/procedure/macro maintenance 11:07am				

Figure 7-128 The Administration menu

Entry forms	Output formats	Search forms	Procedure	Leave
Search forms maintenance 11:07am				

Figure 7-129 The Forms/Procs Maintenance Menu

Entry forms	Output formats	Search forms	Procedure	Leave
		Create/modify Copy Delete Leave		
Create/modify a search form 11:07am				

Figure 7-130 The Search Forms Maintenance Menu

C R E A T E / M O D I F Y S E A R C H F O R M	
Form:	xyz
Description:	
Press ENTER to edit the search form	

Figure 7-131 The Create/Modify Search Form screen

Once you provide a search form name and [optional] description, the first screen of the six page definition form used to generate and modify search forms appears. If you enter a name that has already been assigned (that is, the name of a pre-existing form), you will be allowed to modify it only if you are its author or creator. If you are not, you will receive an error message.

A summary description of the pages and their contents follows.

Search Form Definition

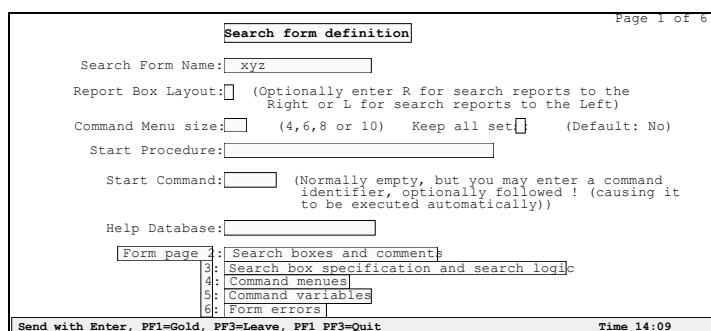


Figure 7–132 Search Form Definition Screens, Page 1

Search Form Name

This field identifies the form; the search form name must be unique.

Report Box Layout

This contains a code (R for 'Right' or L for 'Left') specifying whether the [optional] report window is to be on the right or left.

This item may be left blank if report boxes are not required. If however, a search report window is desired, the search entry window will be seventy columns in width, rather than its customary seventy-eight.

Command Menu Size

This defines the maximum number of items on each page in the command menu window (four, six, eight or ten). However, up to ninety-nine command menu items can be divided among as many menus as needed.

Keep All Sets

If Y for Yes is entered here, all of the search results created while the search form is in use will be kept in CCL's Search History, which might be useful if the results need to be subsequently viewed, saved or edited. However, the maximum number of search results that the search history may contain is set to 255 (a limit that might be reached quickly if the form has many search boxes); therefore the default is set to No. All searches previously performed will then be deleted before the next search is executed.

Start Procedure

This is the name of a procedure to be run as a start-up when the search form is used.

In one of the search forms created for the demonstration database **Alice** called **Alice_Demo2**, the start procedure is a public procedure named **Public.Alice_Demo2_Ini**, the text of which is shown below:

```
!ALICE_SF2_START
!
define f=full
```

A start procedure should be a public procedure (created by the user SYSTEM) or a group procedure created by the manager of the user group that is going to use the search form.

Start Command

This is the identifier of a command item. If specified, the cursor moves to it in the command menu window at startup, and if an exclamation mark [!] follows the identifier, the command is executed. If no identifier is specified, the first search box becomes the default starting point.

Help Database

This is the name of the database where, if desired, the creator of a search form stores suitable help texts for each search box as well as each command menu item.

Each record in the Help database can be assigned to any search box or command item by using its record number. When a user presses <Help>, the relevant record(s) will be presented using the default output format specified in the database design. If no default output format for the Help database has been provided, the Help records will be shown in the system default format.

Form Page

This Table of Contents lists the major topics to be found in the remaining five pages of Search Form Definition.

Below is Page 1 of the Search Form Definition for the **Alice** search form called **Alice_Demo**:

Search form definition

Page 1 of 6

Search Form Name:

Report Box Layout:☐ R (Optionally enter R for search reports to the Right or L for search reports to the Left)

Command Menu size: (4,6,8 or 10) Keep all set☐ (Default: No)

Start Procedure:

Start Command: (Normally empty, but you may enter a command identifier, optionally followed ! (causing it to be executed automatically))

Help Database:

Form page

2

3

4

5

6

Search boxes and comment

Search box specification and search log

Command menus

Command variables

Form errors

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit

Time 14:09

Figure 7–133 Page 1 for Alice_Demo2

Note:
You need to be logged in as SYSTEM, the creator of Alice_Demo to obtain this form.

Search Box Layout and Comments

The second page of the definition form specifies the contents and the layout of the search entry window.




Figure 7–134 Search Form Definition Screens, Page 2

The search form layout area is the same size and shape as the search entry window. Here the position and extent of the search boxes of the form are specified, and comments may be added to identify the boxes and otherwise assist the user of the form.

If no search report windows have been defined, the layout area will measure nineteen lines by seventy-eight columns, so that more than one search box may be defined on the same line and the boxes and/or comments may extend into the eight right- or left-most character positions normally reserved for report boxes.

The start of a box is indicated by the left chevron character [**<**], and its end (on the same line) by the right chevron [**>**]. Enter an identifying number for the box within the chevrons for use in the specification on the next definition page. The identifying number should be an integer between one and ninety-nine or the maximum number of boxes used in the form, whichever is lesser.

Any entry that is not a valid box number surrounded by a matched pair of chevrons will not be recognized as a well-formed search box, and will appear as a comment on the search form. Search boxes and *literals* or fixed texts have absolute positioning; that is, their on-screen location is the same as their position on the form.

At run time, the search form tab order is always sequential by box number, regardless of box layout position. For example, if a cluster of boxes is laid out on the screen in this order:

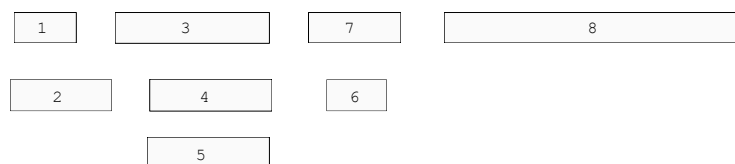


Figure 7–135 Sample search form layout

then the tab order or cursor path would still be one ô two ô three ô four ô five ô six ô seven ô eight, like this:

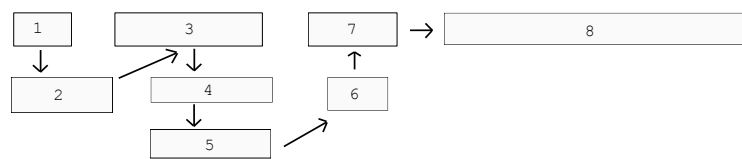


Figure 7–136 Sample layout tab order

Note:
A search box can occupy only one line, and the maximum number of search fields in a form is ninety-nine.

Page 2 of 6

Search box layout and/or comments

*** Advanced Searching in ALICE database ***

Chapter No.: <1 >

Chapter Heading: <2 >

Speaker in text: <3 >

Person in text: <4 >

Any other term(s) in text: <5 >

PLEASE ENTER SEARCH TERM(S) THEN PRESS <ENTER> OR PRESS <PF3> TO LEAVE

Right limit if report boxes are used

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7–137 Page 2 for Alice_Demo2

Search Box Specification and Search Logic

This page contains the search box specifications, search box attributes, search logic to be used and record numbers of search box help texts from the search form Help database.

All of these are *tupled* fields, so to move to the first column of a new line type and/or <Tab> across all the columns until finished with entry for that line, then press ↵.



Id	Box specification	Restriction	ASE	Attribute	Help

Logic: Default help:

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7–138 Search Form Definition Screens, Page 3

ID

This is the identifying number of the box (between chevrons) that was entered in the search form layout screen.

Box Specifications

The box specifications are search fragments, for example

fieldname=*

The asterisk [*] is used to represent individual search expressions, and will be replaced by whatever term or terms the user inserts at run time.

If you provide only one fragment, that same fragment will be used for <Select> (**Display**) and <Enter> for search. If you insert two fragments (separated with commas), the first will be used for <Select> and the second for <Enter>. The maximum is two search fragments.

These actions are governed by *substitution rules*, which control how a term entered into a search box is sent to CCL. Here the *search domain* for each box is defined, including any field names, types, view names, truncation, **Display** vs. **Find** action, indirect searching and exact matching that might be used.

Examples of these for the demonstration database **Alice** are given below:

Sample Substitution Rule	Area of Effect
phrase=*	both Find and Display
chapter=*\$,chapter=(*)	Display , Find
speaker=nt(\$*)	both Find and Display

Table 7-44 Examples of search form substitution rules

Phrase=*

Since this box contains a single expression (rather than two, separated by a comma), it will be used for both **Display** and **Find** instructions.

For example, when the user enters the term 'cat', the command produced will be

Find PHrase=cat

or

Display PHrase=cat

Note:

*If the user leaves the search box empty and presses <Select>, TRIP will perform a **Display PHrase=\$**, causing all field contents to be listed.*

Chapter=*\$,chapter=(*)

The first fragment, **chapter=*\$**, applies only to **Display** orders, and this particular expression specifies that only the **Display** term will be right-truncated.

For example, when the user enters the term 'cat' and presses <**Select**>, the request generated is

Display chapter=cat\$

The second phrase, **chapter=(*)**, applies only to **Find** requests. The parentheses around the **Find** term serve to extend the scope of the field name. If the user enters 'pig **OR** cat' in a box *with* parentheses in its substitution clause, it will be interpreted as '**Find chapter=(pig OR cat)**', while *without* parentheses it would be interpreted as '**Find (chapter=pig) OR cat**'.

For example, when the user enters the expression 'queen or Alice', the order created is

Find chapter=(queen or alice)

Note:

If you need to redefine the wildcard characters, keep in mind that TRIP has a special use for the asterisk [] in search forms. The dollar sign [\$] is preferable to the number or pound sign [#] as a truncation character alternate, since it cannot be redefined.*

Speaker=nt(\$*)

This phrase generates a single-level thesaurus down-search that is left-truncated. The search against the *thesaurus* is truncated, not that against the

source database. The single fragment will be used for both **Display** and **Find** requests.

For example, when the user enters the term 'ice', the command generated is

```
Find speaker=nt($ice)
```

or

```
Display speaker=nt($ice)
```

Restriction

When the number of a preceding search field is included in another search box specification, an 'implicit AND' is created between that prior field and the new search box specification. The effect for both **Display** and **Find** is like including '**S=x**' in a search request. The pool or range of records available to any search conducted using the new specification will then be restricted to those records located by the previous search field. For example, if Search Box 1 finds thirty-one records in a one hundred-record database and is entered into Search Box 2 as a restriction, the maximum number of records Search Box 2 will have to search in is thirty-one (search one *and* two), not one hundred.

This feature is useful for showing the cascading action or cumulative history of commands executed with search report boxes.

ASE

An ASE may be called from a search box. The name given in this column specifies the ASE to be called when a user enters one or more terms and leaves the search box by any method (<Tab>, arrow keys etc.)

The ASE is called after the user exits the box but *before* the search is performed, to allow the ASE to modify the terms being searched. This is useful where conversions are necessary, for example, changing from metric to imperial measurements.

Attribute

You may specify video attributes for search boxes directly in the search form definition, valid attributes being Bold (B), Reverse (R), Underline (U) and Blink (L). Search boxes can have different, as well as a combination of, attributes.

The table following presents all valid combinations of video attributes:

Bold	Reverse	Underline	Blink
✓			
✓		✓	
✓			✓
✓		✓	✓
	✓		
	✓	✓	
	✓		✓
	✓	✓	✓
		✓	
		✓	✓
			✓

Table 7–45 Valid video attribute combinations

Help

When a user presses <Help>, the record specified in the Help column on Page 1 of the Search Form Design will be shown in its default format.

Logic

This specifies how the results of the search boxes are to be combined for the form, and takes the form of a complete search expression with the numbers of search boxes as terms.

AND is assumed if this area is left blank. To create any other relationship, enter the search box numbers joined by **AND**, **OR**, **NOT** or **XOR** as desired. For example:

(1 OR 2) AND (3 OR 4) NOT (1 AND 2)

Default Help

This is the number of a help record to be used for search boxes with nothing in their Help slot.

Page 3 of 6

Id	Box specification	Restriction	ASE	Attribute	Hel
1	Chaptnr= (*)			R	1
2	chapter= (*)			B	1
3	speaker= (*#), speaker= (*)			B	1
4	person= (*#), person= (*)			B	1
5	(*)			R	1

Logic: Default help:

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7–139 Page 3 of Alice_Demo2

Command Box Specification

The fourth page defines the set of commands to be used with the form. There are ten fields, and the six fields below the default specifications (described later) are scrolled in parallel, with each line specifying a command.

Command box specification

Defaults

Command	Description	Id.	Nex.	Exc.	Help
---------	-------------	-----	------	------	------

Command box to be activated after a successful search:

Page 4 of 6

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7–140 Search Form Definition Screens, Page 4

Command Description

This is a short text indicating the nature of the command to be applied to the current search box, e.g. ‘Show hits’, ‘Scroll forward’, ‘Help’, ‘Next record’, ‘New search’ etc. This and nothing else appears in the menus on the form.

Command

These are of four types: CCL instructions, *command variables*, *option variables* and *search form commands*, and are defined according to their use in *option* or *command menus*.

An option menu is used to generate CCL fragments to be used by another command menu button, such as ‘Format’ in a predefined **Show** order. A command menu is used to produce alternatives from complete CCL instructions to be encoded on the form, for example, ‘Choose Database’.

CCL Commands

These are either complete CCL commands, i.e. **Show**, **More**, **Back** etc., or instructions where part of the command will be replaced using a substitution rule (see the section on substitution rules for more information). For example,

Show Format=<1> ↵

where the value of <1> is defined by an option value declaration elsewhere in the form (see the next section).

In the case of command and option variables, the command produces a pop-up menu offering several options to the user.

Form variable numbers must be positive integers between one and ninety-nine, and they are used to inspect and/or change the value of the variables.

Option Variables

These allow a form designer to define a placeholder in a CCL instruction. This must be a positive integer between one and ninety-nine, and is used to uniquely identify a pop-up menu in the 'Command' column on Page 5 of the 'Search Form Definition Screens'.

Command Variables

These are similar to option variables, but rather than being composed of a fragment of a CCL command, each contains an entire CCL instruction. To create a command variable, place an exclamation mark [!] immediately after the positive command integer, which will identify it as destined to appear on a command menu rather than on an option menu.

Search Form Commands

Recognized search form commands are -2, -1, and 0:

- 0 returns the cursor to the search entry window leaving its data intact. It is used to modify searches.
- -1 switches over to normal CCL command mode searching (<Leave> switches back to the form).
- -2 clears the search window and moves the cursor to its first search box, and is thus used to begin a new search.

ID

This command identifier consists of the number of the menu followed by a full stop and the number of the command within the menu (0-9), for example, 1.8.

Next

This command identifier specifies where processing is to continue if the current command ends *normally*. For example, a command such as 'Scroll forward' ends normally as long as there is more to show. An exclamation mark after the command identifier will result in the command being executed immediately.

If the command box is left blank, the 'Next' option will always be acted upon immediately after. This is useful for chaining commands together. For example, command number 1.3 may be connected to 1.4, causing its

activation, which leads to the execution of 1.5, and breaking the chain before performance of 1.6.

Exception

This is an identifier specifying where processing is to continue if the current command ends *abnormally*, i.e. with an exception. A command such as 'Scroll forward' ends abnormally when there is nothing more to show. An exclamation mark may be used here in the same way as for 'Next' commands.

Help

This is the number of a record from the Help database that explains the command on the line. This record appears on screen when a user presses the <Help> key.

The Defaults

The three unlabeled boxes below the screen header 'Defaults' are, from left to right, 'Default Next', 'Default Exception' and 'Default Help', as shown below (text in default boxes has been added for clarity):

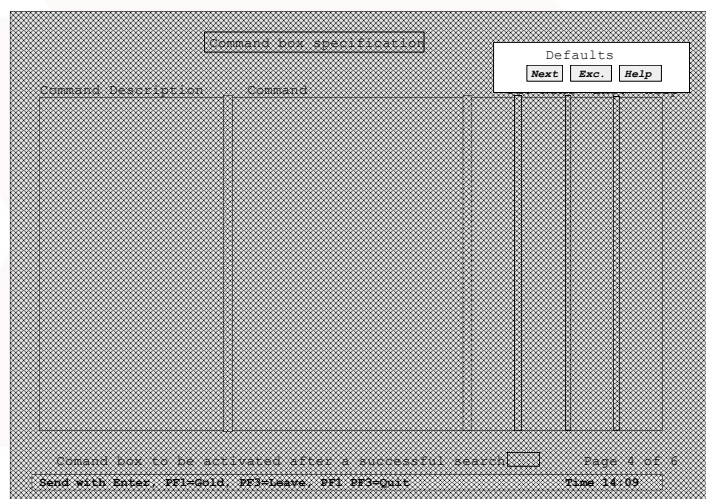


Figure 7-141 The Next, Exception and Help defaults

Default Next

This is a command identifier to be used when a command is specified without any 'Next' value.

Default Exception

This is a command identifier to be used when a command is specified without any 'Exc.' value.

Default Help

This is the number of a help record to be used when a command is specified without any 'Help' value.

Command Box Activated After a Successful Search

This disables a specific ‘Next’ action for a new search. When a user completes a search, any command given here will be activated rather than any that may have been provided in the ‘Next’ column.

Command box specification		Defaults			
Command Description	Command			1	
		Id.	Next	Exc.	Help
New Search	-2	1.0	1.1		1
SHOW Menu		1.1	2.0!	1.4	1
PRINT Options	1!	1.2	1.0	1.4	1
Leave TRIP	stop	1.3			1
Modify Search	0	1.4	1.1	1.0	1
Change Format	2!	1.5	1.1!	1.1	1
Goto CCL	-1	1.6	1.0	1.7	1
Database Status	status	1.7			1
Show Results	show	2.0	2.1	1.4	1
Scroll Forwards	more	2.1	2.1	2.5	1
Next Record	next	2.2	2.2	2.6	1
MAIN Menu		2.3	1.1	1.0	1
Select Record	run_alice_demo2_sel	2.4	2.0!	2.0	1
Scroll Backwards	back	2.5	2.5	2.1	1
Previous Record	previous	2.6	2.6	2.2	1
Sort Output	3!	2.7	2.1	2.7	1

Command box to be activated after a successful search: 1.1

Page 4 of 6

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit

Time 14:09

Figure 7–142 Page 4 of Alice_Demo2

Command Variables

This page defines the form variables as well as their legal values. The first value specified for a variable becomes its initial default. There are three fields on the page and they are scrolled in parallel with one form variable value per line. A maximum of 99 search boxes may be included on any single search form; however, no more than 30 of the boxes may contain values when the search is executed.

Command Variables			
			Page 5 of 6
Id	Value description	Legal Values	
1	Print to file	print s=0 no hold file=printfile.out	
1	Print and await compl@	print s=0 wait	
1	Print upon leaving TRIP	print s=0 hold	
2	Short Format	define format=short	
2	Full Format	define format=full	
3	Sort on Chapter Heading	s sort=chapter	
3	Sort on Speaker	s sort=speaker	
3	Sort on Person	s sort=person	

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit

Time 14:09

Figure 7–143 Search Form Definition Screens, Page 5

ID

This is the form variable identifier, which must be a positive integer between 1 and 99. These were included on Page 4 of the Search Form Definition Screens for option and command menu variable definitions.

The system reads this page whenever a user inserts a search expression into a box containing a variable declaration and presses ↵. For example, if a

variable declaration was '1', the system will produce a pop-up menu with all the possible values of '1' listed on it.

Value Description

This is a text describing the effect of selecting the value given on the same line in the next field, which appears as an item on a pop-up menu.

Legal Values

If an item will appear as an option menu, 'Legal Value' will be a fragment. If an item is to appear on a command menu, 'Legal Value' will be a full CCL command.

An exclamation mark after the variable number in the command box specification on Page 4 will cause the variable command to be executed immediately on selection.



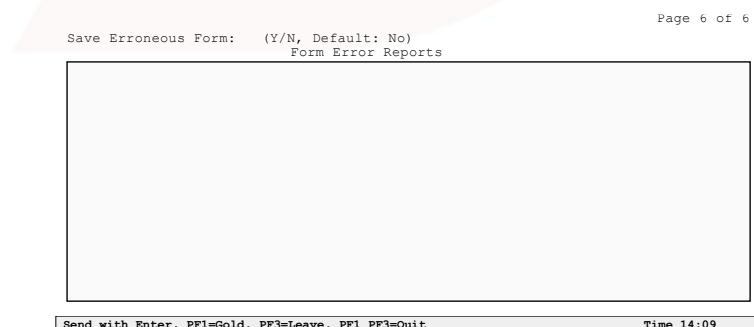
Id	Value description	Legal Values
1	Print to file	print s=0 no hold file=printfile.out

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7-144 Page 5 of Alice_Demo2

Form Errors

When the search form is saved, any errors that may have been made in the specification of the search form are displayed on the last page, e.g. use of a non-unique command identifier or non-existent Help database.



Save Erroneous Form: (Y/N, Default: No)

Form Error Reports

Send with Enter, PF1=Gold, PF3=Leave, PF1 PF3=Quit Time 14:09

Figure 7-145 Search Form Definition Screens, Page 6

The form can be saved with any listed errors by specifying Y for Yes in the 'Save Erroneous Form' box. It is not possible to use an erroneous search form.

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 278 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 **Highlight** 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	<i>INSert, UPDate or DElete</i>	<i>Field, Subfield, Word, Sentence etc.</i>	<i>XYZ, 123, etc.</i>	<i>WHere R=521, 687, 688, 1023, 2190, 2349 or WHere S=3</i>

Table 8–46 Anatomy of a global update command

As with all CCL commands, the *maximum expanded order length* is 400 characters. A longer order will lead to an error message.

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	aaabbb cc 112233	<i>R=record number</i>

Table 8–47 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.

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 8–48 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 **TEExt** 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:

1. **TO** a number
2. a number **TO** a number
3. **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**, **DELe**te 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 (<i>for use with DELeTe only</i>) Field Subfield Paragraph Sentence Word Fieldname.Subfield Fieldname.Paragraph Fieldname.Paragraph.Sentence Part.Fieldname.Subfield Part.Fieldname.Paragraph Part.Fieldname.Paragraph.Sentence	aaabbbccc 111222333	S= <i>search number</i>

Table 8–49 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
FIEld	FIE
SUBField	SUBF
PARagraph	PAR
SENtence	SEN
WORD	WORD

Table 8–50 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 **TEText** field **content** from the records in search number four.

Example 2:

```
DELeTe content.4 WH S=4
```

deletes the fourth paragraph of the **TEText** 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:

1. If the word hit consists of upper-case letters only, the string will be converted to upper-case before the exchange.

2. 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 *GU`database`name unique ID.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 **TEText** 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 **TEText** 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

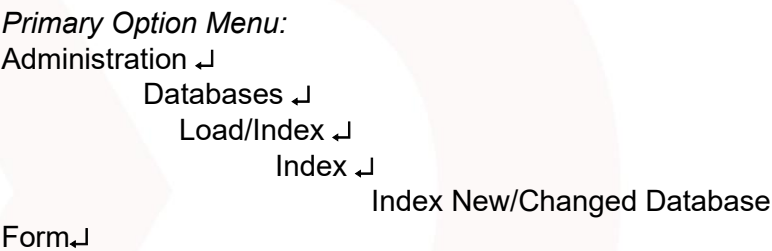
TRIP provides three utilities for automated data entry (loading) and database indexing (making the data searchable), 'Index', 'Load/Index' and 'Load'. All three utilities submit jobs to be executed either in background mode.

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' (as opposed to 'Load/Index') if you are using manual data entry to write information to the **BAF**.

Follow the pathway diagrammed below to access 'Index'.



This pathway brings you through the following series of screens:

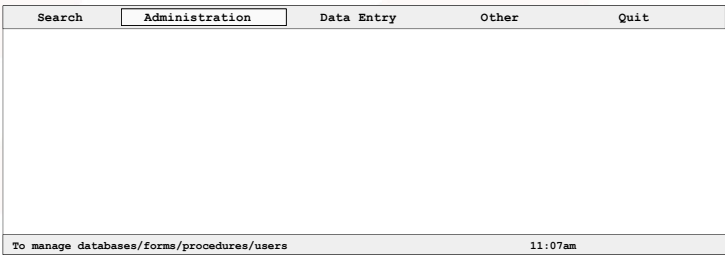


Figure 9–146 The Primary Option Menu

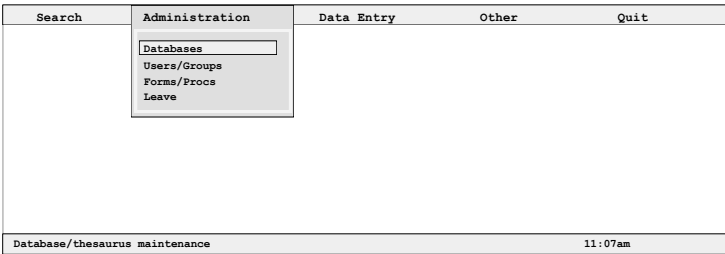


Figure 9–147 The Administration menu



Figure 9–148 The Load/Index menu

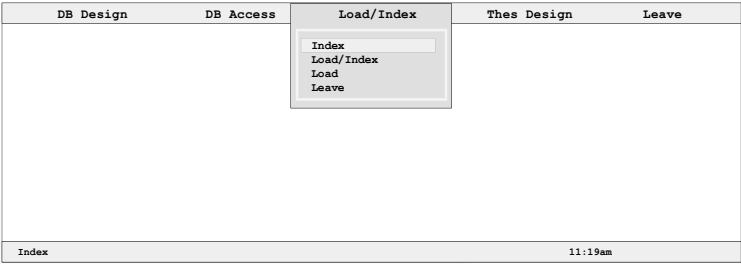


Figure 9–149 The Index option

INDEX NEW / CHANGED DATABASE

Database:

Figure 9–150 The Index New/Changed Database form

The command **INDeX** may be used instead of the screen form. For example, the order:

INDeX corr

given in the CCL command window or as part of a TRIP procedure begins a process indexing the database **Corr**.

Load/Index

With 'Load/Index', automated data loading to the **BAF** is immediately followed by the updating (indexing) of the **BIF** and **VIF**.

Choose this option (as opposed to 'Index') when all data to be written to the database is contained in a **TForm** file.

Follow this pathway for 'Load/Index':

Primary Option Menu:

Administration ↵

Databases ↵

Load/Index ↵

Load/Index ↵

Load and Index Database Form ↵

This pathway brings you to these screens:

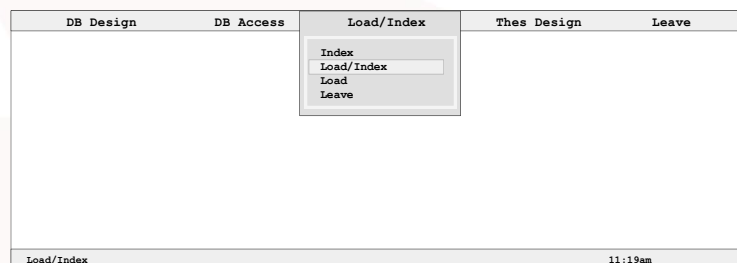


Figure 9–151 The Load/Index option

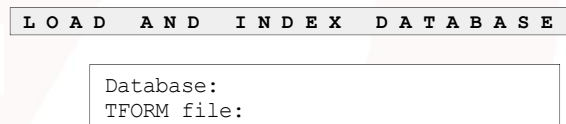


Figure 9–152 The Load and Index Database form

Load

Loading uses records from a file in TRIP's entry format **TForm** to update the **BAF**.

Follow the pathway diagrammed below to access 'Load'.

Primary Option Menu:

Administration ↵

Databases ↵

Load/Index ↵

Load ↵

Load Database Form ↵

This brings you to these screens:

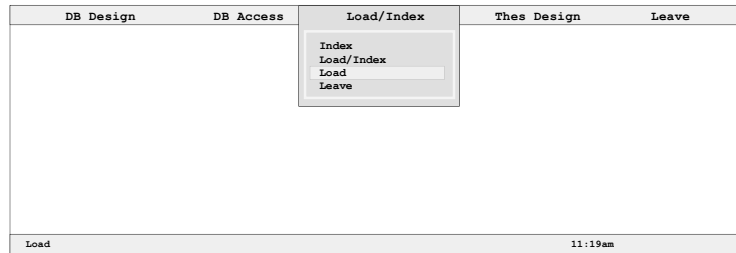


Figure 9–153 The Load option

L O A D D A T A B A S E

Database:
TFORM file:

Figure 9–154 The Load Database form

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 9–51 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 environment variable TDBS_LOG.

Note:

In order to conserve disk space, it is important to remember to delete the log files, or set up a regular 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 environment variable TDBS_LOG. If TDBS_LOG are undefined, then ERRLOG_databasename.TFO will be stored in the current working 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 must be removed by running a utility called BAFINI in this manner:

Operating System	Command
UNIX	\$TDBS_EXE/bafini
Windows	bafini

Table 9–52 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 referenskölla. (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 Privilege

Access Privileges

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 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".

The 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 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

The number of possible users depends on the system site license purchased.

Creating a New User

To create a new user, follow the pathway below:

Primary Option Menu:

Administration ↵

Users/Groups ↵

User ↵

New User ↵

Create New User Form ↵

This pathway also leads to the 'Delete User', 'Profile' and 'Change UM' user options.

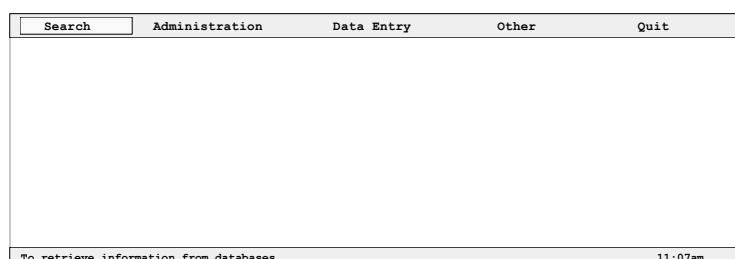


Figure 10–155 **The Primary Option Menu**

Search	Administration	Data Entry	Other	Quit
	Databases Users/Groups Forms/Procs Leave			
User/group organisation 11:07am				

Figure 10–156 The Administration menu

User	Group	System	Leave
Organising individual access to the system 11:07am			

Figure 10–157 The User option

User	Group	System	Leave
New user Delete user Profile Change UM Leave			
Introducing a new user 11:07am			

Figure 10–158 The New User option

CREATE NEW USER

Username:
Password:
Verification:

Figure 10–159 The Create New User form

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.

Send the form by pressing ↵. 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 User

To obtain the 'Delete User' screen, follow the pathway as for 'New User' to the 'User' option, then choose 'Delete User' and press ↵.

User	Group	System	Leave
New user <input type="button" value="Delete user"/> <input type="button" value="Profile"/> <input type="button" value="Change UM"/> <input type="button" value="Leave"/>			

Delete a user 11:07am

Figure 10–160 The Delete User option

D E L E T E U S E R

Username:

Figure 10–161 The Delete User form

Note:

When a user is deleted, all database access rights and privileges for that user are removed.

You must transfer all of a database administrator's or user manager's holdings before that manager can be deleted, and only the creator of a user can delete that user.

The user SYSTEM cannot be deleted.

User Profiles

To obtain the 'User Profile' screen, follow the pathway as for 'New User' to the 'User' option and choose 'Profile', then press ↵.

User	Group	System	Leave
New user <input type="button" value="Delete user"/> <input type="button" value="Profile"/> <input type="button" value="Change UM"/> <input type="button" value="Leave"/>			

To change user profile 11:07am

Figure 10–162 The Profile option

U S E R P R O F I L E

Username:

Full name:

Company:

Address:

Phone:

Start proc:

Start module:

User manager:

File manager:

Group member:

Enter without password if O/S username = TRIP username: ☐

Date form: for May 1st, 1988

Figure 10–163 The User Profile form

The user manager who has created a user identity may enter data such as name and address and some TRIP defaults in the user's profile.

Selecting the start module 'CCL Search' deposits the user directly in the search order window after login, while 'Form Search' opens a search form. Make a selection here by placing the cursor beside the desired option and pressing **<Select>**.

Providing the name of a start procedure causes the TRIP procedure specified to run by default each time the user logs in.

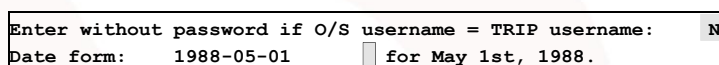
Local system validation

A user may enter TRIP without password if he or she has the same O/S user name and TRIP user name. Place the cursor beside 'TRIP username:' on the line beginning with 'Enter without password ...' and enter a 'Y' for Yes after the prompt to grant this privilege.

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, position the cursor to the immediate left of 'for May 1st, 1988'.

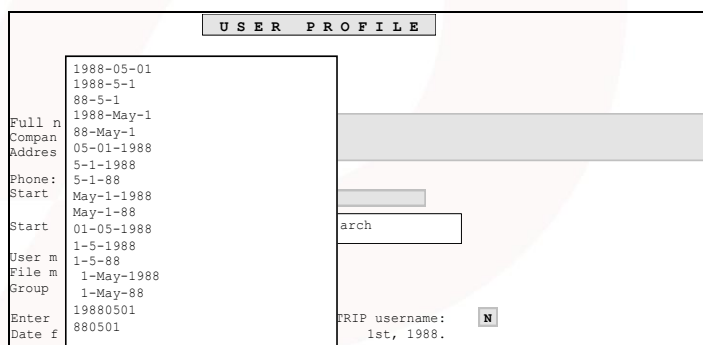


```

Enter without password if O/S username = TRIP username: N
Date form: 1988-05-01 for May 1st, 1988.
  
```

Figure 10-164 Changing the Date Format

and press **<kp 9>** for a list of the available formats:



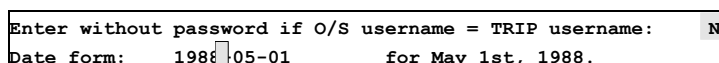
```

USER PROFILE
1988-05-01
1988-5-1
88-5-1
1988-May-1
88-May-1
05-01-1988
5-1-1988
5-1-88
May-1-1988
May-1-88
01-05-1988
1-5-1988
1-5-88
1-May-1988
1-May-88
19880501
880501
TRIP username: N
  
```

Figure 10-165 The Date Form box

Move up and down the list with the **<Arrow keys>** and choose whatever form you like with **<Select>** (the default format is the first on the list, 1988-05-01). You will be returned to the User Profile window.

To change the digit separator characters, use **<Tab>** or the **<↑ Arrow>** or **<↓ Arrow>** keys to move to the first separator:



```

Enter without password if O/S username = TRIP username: N
Date form: 1988 05-01 for May 1st, 1988.
  
```

Figure 10-166 Changing the first date separator

and overwrite that character with a hyphen [-], slash [/], full stop [.], or full colon [:].

Repeat with the second separator:

Enter without password if O/S username = TRIP username:	N
Date form: 1988-01-01	for May 1st, 1988.

Figure 10–167 Changing the second date separator

To alter either the order of the elements in the date or the manner in which the month or day is expressed, you must choose a date form from a list obtained by placing the cursor in the last position on the date form line and pressing **<Gold><List>** (**<Gold><kp 9>**). Choose any one of the seventeen alternatives by placing the cursor beside one and pressing **<Select>**.

The current date form is recognized in search orders (in addition to the system default), and output by the system in **STatus** lists and in the output of records. An output format may, however, specify some other date form.

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.

To display the 'Transfer User Managership' screen, follow the pathway as for 'New User' to the 'User' option, choose 'Change UM' and press ↵.

User	Group	System	Leave
New user			
Delete user			
Profile			
Change UM			
Leave			

Change the ownership of users/groups 11:07am

Figure 10–168 The Change UM option

T R A N S F E R U S E R M A N A G E R S H I P
Current manager:
New manager:
Transfer object:

An * object entry will transfer managership of ALL users created by current manager.

Figure 10–169 The Transfer User Managership form

Enter the name of the user or user group that is to be transferred and the name of the new manager. If you enter an asterisk [*] instead of the name of a user or group, all identities you have created will be transferred.

The system manager may transfer any user or group to another user manager, regardless of their owners/creators, except for the group PUBLIC.

Creating a User Group

To create a user group, follow the pathway illustrated:

Primary Option Menu:
Administration ↵

Users/Groups ↵
Group ↵
New Group ↵
Create New Group Form ↵

This pathway also leads to the 'Delete Group', 'Add Member' and 'Delete Member' user options.

User	Group	System	Leave
Introducing a new group			

Figure 10–170 The Group option

User	Group	System	Leave
Introducing a new group			

Figure 10–171 The New Group option

C R E A T E N E W G R O U P

Group name:

Figure 10–172 The Create New Group form

Deleting a User Group

To bring up the 'Delete Group' screen, follow the pathway as for 'New Group' to the 'Group' option, choose 'Delete Group' and press ↵.

User	Group	System	Leave
Deleting a group			

Figure 10–173 The Delete Group option

D E L E T E G R O U P

Group name:

Figure 10–174 The Delete Group form

Adding a Group Member

To obtain the 'Add Group Member' screen, follow the pathway as for 'New Group' to the 'Group' option, choose 'Add Member' and press ↵.

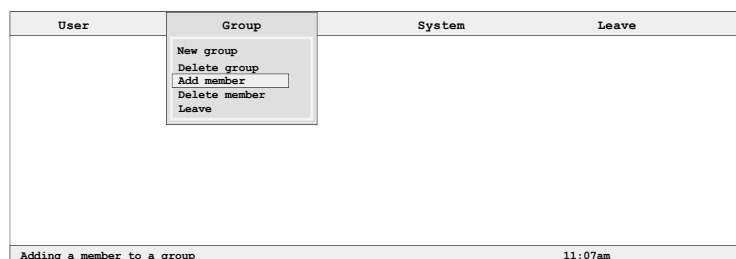


Figure 10–175 The Add Member option

A D D G R O U P M E M B E R

Group name:
New member:

Figure 10–176 The Add Group Member form

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

To obtain the 'Delete Group Member' screen, follow the pathway as for 'New Group' to the 'Group' option, choose 'Delete Member' and press ↵.

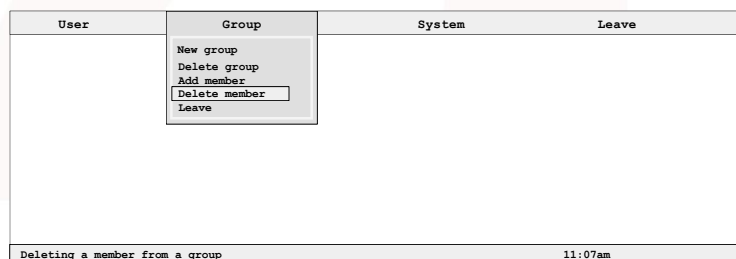


Figure 10–177 The Delete Member option

D E L E T E G R O U P M E M B E R

Group name:
Member:

Figure 10–178 The Delete Group Member form

System Manager Privileges

The system manager has a separate menu for those things that can be done by the system manager only.

To grant manager rights, follow the pathway below:

Primary Option Menu:
Administration ↵
Users/Groups ↵

System ↵
Manager Rights ↵
Grant User/File Manager Rights
Form ↵

This pathway also leads to the 'Set Password' system option.

User	Group	System	Leave
System manager commands			11:07am

Figure 10–179 The System option

User	Group	System	Leave
Manager Rights			11:07am

Figure 10–180 The Manager Rights option

GRANT USER/FILE MANAGER RIGHTS
Username: <input type="text"/>
User manager (Y/N):
File manager (Y/N):

Press ENTER to store new manager rights.

Figure 10–181 The Grant User/File Manager Rights form

Enter the name of the user who is to be given user or file manager (database administrator) rights beside 'Username:', and a 'Y' for Yes beside 'User manager' and/or 'File manager' to assign these privileges. Press <Enter> to store the form.

When SYSTEM grants a user User Manager rights, that user will automatically be transferred to SYSTEM.

Manager rights may be removed by inserting 'N' for 'Y' in each box necessary. Before that is done, however, his or her users and/or groups must be transferred to another manager. Either SYSTEM or the manager may do this using the 'Change FM' and 'Change UM' forms.

Setting a New Password

To access the 'Set Password' screen, follow the pathway as for 'Manager Rights' to the 'System' option, choose 'Password' and press ↵.

User	Group	System	Leave
		<div> Manager Rights <input type="text"/> Password <input type="text"/> Leave <input type="text"/> </div>	
Set new password		11:07am	

Figure 10–182 The Password option

S E T P A S S W O R D

Username:

Password:

Verification:

Figure 10–183 The Set Password form

In times of emergency (for example, when a user has forgotten his or her password), the system manager may assign a new TRIP password using this form. To change your own password, use the 'Password' option under 'Other' on the Primary Option menu.

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 him- or herself, 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, follow this pathway to the 'Database Access Rights Definition' screen:

Primary Option Menu:

Administration ↵

Databases ↵

DB Access ↵

DB Access ↵

Database Access Rights Definition

Form ↵

The same screen form is used to grant access to a database to both a single and a group of users. Beginning at the Primary Option Menu,

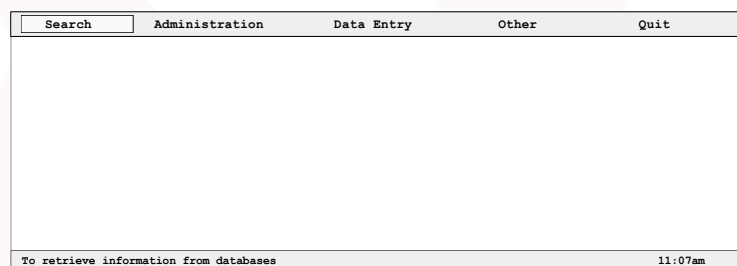


Figure 11–184 The Primary Option Menu

the cursor will be positioned in the upper left corner of the menu, with **Search** bolded or highlighted.

From **Search**, move the cursor right one cell using the <→ **Arrow**> key, and press ↵ or the <↓ **Arrow**> key. The *Administration* drop-down or submenu appears, with *Databases* the default cursor position.

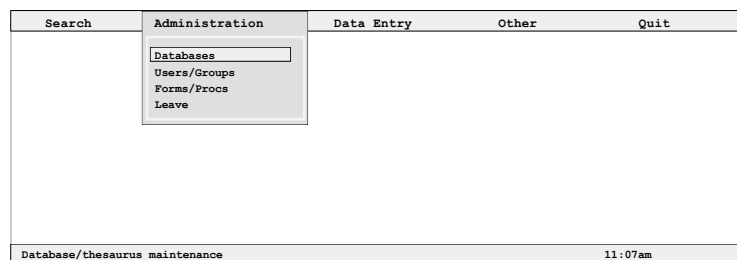


Figure 11–185 The Administration menu

If you wish to return to the Primary Option Menu at this point, press <Leave>, or <Gold><Leave> to quit TRIP altogether.

Press **↓** to choose the *Databases* option, which leads to a new menu headed by the cursor default *DB Design*. Cursor right one position to *DB Access*.



Figure 11–186 The DB Design option

Press **↓** or **<↓ Arrow>** for the *DB Access* option list. Again, pressing **<Leave>** will return you to the Primary Option Menu; **<Gold><Leave>** exits TRIP.

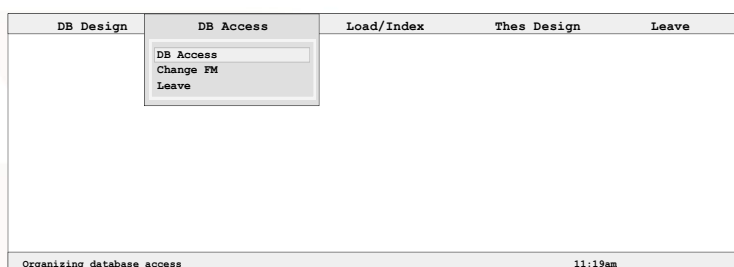


Figure 11–187 The DB Access option

Select the cursor default alternative *DB Access* with **↓** to display the form for assigning access rights.

The first form you will need to fill asks for the name of the database to be accessed and the name of the user or user group requiring access.

D A T A B A S E A C C E S S R I G H T S D E F I N I T I O N
<div> <div>Database:</div> <div>User/group:</div> </div>

Figure 11–188 The Database Access Rights Definition form

If you have just created or modified a database via the *DB Design* menu, the database name will be filled in on the access form. If not, you will need to fill in the name of the database and the name of the user or group. Move from line to line by pressing **<Tab>** or the **<↓ Arrow>** key and press **↓** to register your request. The system checks that you are the owner of the database and that the user or the group exists before displaying the bottom part of the screen.

Note:

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

General Field Access

A user may be assigned almost any combination of the following access rights (see note below):

Type of Access	Effect in Database
READ - All	read access to all fields
READ - None	no read access
READ - Selected	read access to chosen fields
Read Scope	read access to chosen records
WRITE - All	write access to all fields
WRITE - None	no write access
WRITE - Selected	write access to chosen fields
Write Scope	write access to chosen records

Table 11–53 General field access rights

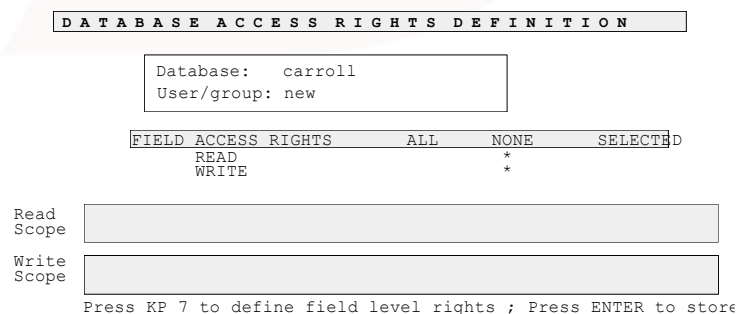
The following combinations are not possible:

Read Access	Write Access
None	All
None	Selected
Selected	All

Table 11–54 Unsupported combinations of access rights

The access form always shows the current state of the user's rights to the database, and contains asterisks [*] in the appropriate columns.

The access rights for a newly-created user are '**READ - NONE**' and '**WRITE - NONE**' by default:



DATABASE ACCESS RIGHTS DEFINITION

Database: carroll
User/group: new

FIELD ACCESS RIGHTS	ALL	NONE	SELECTED
READ		*	
WRITE		*	

Read Scope: []

Write Scope: []

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–189 Default access rights for user 'New'

To change these to 'ALL' or 'SELECTED', move the cursor to the access line and column you wish to assign to this user and press **<Select>**, which moves the asterisk [*] to that column.

Note:

A dash [-] in any 'Field Access Rights' column indicates that this option is non-selectable.

If you have marked the 'SELECTED' column for READ, WRITE or both, position the cursor anywhere on the READ or WRITE access lines and press **<Next Page>** (**<kp 7>**). This displays the 'Database Access Rights Overlay Form', where you can assign the user *per-field* access rights.

Field-Level Access

Here, any or all fields and their contents can be hidden from view or protected from alteration by any or all users. Each field in the database occupies a separate line on the 'Database Access Rights Overlay' form, along with columns for READ and WRITE access. The appearance of the default form varies according to the READ and WRITE assignments on the 'Database Access Rights Definition Form'. The possibilities are summarized below:

If ...	And ...	Then ...	And ...
READ = All	WRITE = Selected	All READ cells are selected	No WRITE cells are selected
READ = Selected	WRITE = None	No READ cells are selected	Selection of WRITE cells not possible (these are shown barred with dashes)
READ = Selected	WRITE = Selected	No READ cells are selected	No WRITE cells are selected

Table 11–55 General field access rights

For example, let us assume that a database administrator has assigned a new user the first option in the table above, involving total read access and selected write access to the database **Carroll** as shown below:

DATABASE ACCESS RIGHTS DEFINITION

Database: carroll
 User/group: new

FIELD ACCESS RIGHTS	ALL	NONE	SELECTED
READ	*	-	*
WRITE			

Read Scope
 Write Scope

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–190 Access to Carroll for user 'New'

The default 'Database Access Rights Overlay' will appear with all fields defined as readable by this user and none as modifiable:

DATABASE ACCESS RIGHTS DEFINITION

Database: carroll
 User/group: new

FIELD NAME	READ	WRITE
CHAPTER	*	
CHAPTNR	*	
PERSON	*	
SPEAKER	*	
TXT	*	
VERSE	*	
TXT2	*	
BOOK	*	

Read Scope
 Write Scope

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–191 The default Access Rights Definition overlay

Assign write access (read and/or write access if you have chosen 'SELECTED' read access) by positioning the cursor in the field you want to grant access to and press **<Select>**. An asterisk will appear in that line and column. To deselect an option (remove an asterisk), use **<Gold><Select>**.

DATABASE ACCESS RIGHTS DEFINITION

Database: carroll
 User/group: new

FIELD NAME	READ	WRITE
CHAPTER	*	*
CHAPTNR	*	*
PERSON	*	
SPEAKER	*	
TXT	*	
VERSE	*	*
TXT2	*	
BOOK	*	*

Read Scope
 Write Scope

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–192 Field Access Rights defined for user 'New'

If the database administrator does not select *field WRITE* access for at least one field on this screen, the *database WRITE* access selection on the previous screen will automatically revert to 'WRITE - NONE'.

When finished, press **<Enter>** to save this portion of the form and return to the 'Database Access Rights Definition Form'. If no further modifications are necessary, press **<Enter>** once more to commit the entire form.

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, entering this command fragment in 'Read Scope' allows this user to read only those records containing the terms 'walrus' and/or 'carpenter' (a *positive* read scope):

walrus OR carpenter

and entering this command in 'Write Scope' allows modification of only those records containing 'walrus' or 'carpenter' and 'Alice':

walrus OR (carpenter AND alice)

as shown in the form below:

DATABASE ACCESS RIGHTS DEFINITION				
Database: carroll				
User/group: new				
FIELD	ACCESS RIGHTS	ALL	NONE	SELECTED
	READ	*	-	*
	WRITE			
Read Scope	walrus or carpenter			
Write Scope	walrus or (carpenter and alice)			

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–193 Record-level access rights for user ‘New’

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**:

ALL NOT (walrus OR carpenter)

as shown below:

DATABASE ACCESS RIGHTS DEFINITION				
Database: carroll				
User/group: new				
FIELD	ACCESS RIGHTS	ALL	NONE	SELECTED
	READ	*	-	*
	WRITE			
Read Scope	all not (walrus or carpenter)			
Write Scope	all not (walrus or (carpenter and alice))			

Press KP 7 to define field level rights ; Press ENTER to store

Figure 11–194 New record-level access rights for user ‘New’

You may restrict write access to selected records independently of record-level read access.

When you have completed the forms for this user, press <Enter> to save. The user’s access rights to the database will be stored in that user’s profile.

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 SScope** 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 **View**, 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 **Show**, a **Print** or a **Print Local** order. The predefined output formats 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 output formats.

A database administrator should keep in mind that users may have limited read access to a database when designing output formats.

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, follow this pathway to the 'Transfer File Managership' screen:

Primary Option Menu:

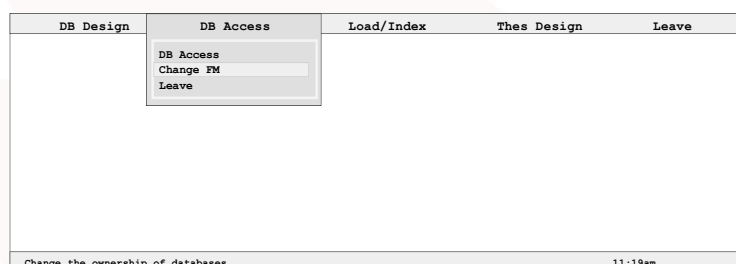
Administration ↵

Databases ↵

DB Access ↵

Change FM ↵

Transfer File Managership Form ↵

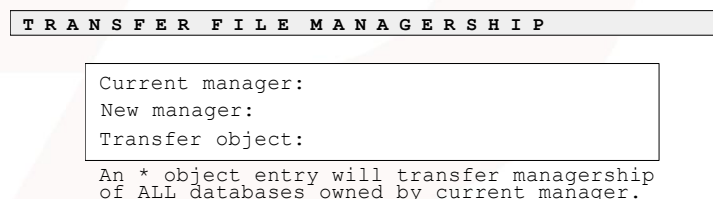


DB Design	DB Access	Load/Index	Thes Design	Leave
	DB Access Change FM Leave			

Change the ownership of databases 11:19am

Figure 11–195 The Change FM (File Manager) option

Select *Change FM* with ↵ to display the form for transferring file managership.



T R A N S F E R F I L E M A N A G E R S H I P
Current manager: New manager: Transfer object: An * object entry will transfer managership of ALL databases owned by current manager.

Figure 11–196 The Transfer File Managership form

Enter the name of the database to be transferred beside 'Transfer object:' and the name of the new database administrator next to 'New manager:'. To transfer the file manager rights for all of your own databases, enter an asterisk [*] on the line beside 'Transfer object:'.

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:

```
(s ac r=carroll)
```

Databas: CARROLL		Owner: SYSTEM
User/Group	Read access	Write access
SYSTEM	ALL	ALL
JOHNSON	ALL	NONE
BRADLEY	ALL	NONE
SMITHE	ALL	ALL
PUBLIC	ALL	ALL

(Enter your command, please.) (Database)

Send with Return, PF1=Gold, PF2=Help, PF3=Leave, PF1 PF3=Quit 11:39am

Figure 11–197 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 is supporting the Euro symbol and that it can define the host character set to be Windows Latin-1 (CP1252) or Windows EE (CP1250, Estonian).

The implementation of the Euro in TRIPclassic has been tested under Reflection V10.x from WRQ, using the above settings.

Note:

If your host character set is incorrectly defined to be DEC supplemental or Latin-1 (ISO 8859-1), then Reflection 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 not be regarded as a Euro symbol when presented in a TRIP client application or when running Reflection with the correct Euro symbol settings. So, ensure the terminal emulator is correctly set-up before entering the Euro symbol into TRIP.

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 TRIPnpx 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 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 opening many databases may not exceed this length.*
2. *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'.*
3. *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.*
4. *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

Keyboard Key Combinations for Emulating VT Keypad Keys

TRIPclassic was developed for use on DEC VT terminals, hence it has inherited the key mappings for an extended VT keyboard. As some keyboards do not have all keypad keys mapped (particularly those on laptops) TRIPclassic incorporates a mechanism whereby certain keyboard key combinations emulate the keypad keys. These combinations are as follows:

Emulate <PF1>, <PF2>, <PF3> and <PF4>:

- Press <Ctrl> and <F> together, then press the respective number key, i.e. <1> to <4>.

Emulate the keypad number keys:

- Press <Ctrl> and <K> together, then press the respective number key, i.e. <1> to <9>.

Emulate the keypad keys, < . >, < , > and < - >

- Press <Ctrl> and <K> together, then press the respective keys, i.e. < . >, < , > and < - >.

Emulate the keys <Page Up>, <Page Down>, <Backspace> and keypad <Enter>

- For <Page Up> (or <Next page>), press <Ctrl> and <N> together.
- For <Page Down> (or <Prev page>), press <Ctrl> and <P> together.
- For <Backspace>, press <Ctrl> and together.
- For Keypad <Enter>, press <Ctrl> and <e> together.

Should it become necessary to remap a terminal emulation's keys, then the above key combinations should suffice.

Note:

In the TRIPclassic for windows interface, when using a standard 102 key keyboard, the VT keypad function keys <PF1> to <PF4> are transposed to the PC function keys <F1> to <F4> respectively.

Obtaining Version and License Information

TRIPsystem Version Information

The TRIP system current version number can either be ascertained by entering the following command at the command prompt:

```
trip -v
```

which will immediately return the version number, thus:

```
7.0-0:3
```

or by simply starting a TRIP session and viewing the logon screen. The current version number is displayed below the 'T R I P' title line:



Figure B-1 TRIPsystem version information

If it is not required to log into TRIP at this time, press the <PF3> key (or <*> on a PC keypad) to exit TRIP.

Note:

See the “CCL and the TRIPclassic Numeric Keypad” section on page 10 of this guide for a diagram of VT to PC keypad mappings.

Updating a TRIP Product License Key

To install or update a TRIP product license key, it is necessary to run the SETLOCK program. This is done:

- On UNIX, from a command prompt, by changing to the TRIP installation's *bin* directory and entering command *./setlock*.
- On windows, by starting a command Window (i.e. Entering *cmd.exe* from the Start menu search box) and then entering the command *setlock* at the prompt.

The setlock program will then prompt you to enter the license key details that you have received from your TRIP distributor.

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 which is 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).*

Configuring LDAP

The following section explains how to configure TRIP to use an LDAP repository for authentication; this requires editing the [Privileged] section of tdb.conf

Note:

LDAP variables are only ever valid in the [Privileged] section of tdb.conf

By default, when using an external authentication provider such as LDAP, if a user provides a valid set of credentials for that authentication provider and the user is unknown to TRIP, the user will be logged into TRIP as a guest user (under the BUILTIN_GUEST account). To disable this functionality set the following variable:

```
TDBS_DISALLOW_GUEST=True
```

To establish LDAP as the authentication provider, set the following variable:

```
TDBS_AUTH_PROVIDER=LDAP
```

The default behaviour of the system in the absence of such a setting is to fallback to using CONTROL for all authentication requests.

```
TDBS_AUTH_PROVIDER=LDAP
```

The LDAP provider needs to know which servers are capable of authenticating. The following variable definition can be a single server, or can be a list of servers, each of which can optionally state a port number. For example:

```
TDBS_LDAP_SERVER=server1, server2:3030, server3
```

In the absence of port numbers, the default port for LDAP (or LDAP over SSL) will be provided by the system. `TDBS_LDAP_SERVER=pluto`

Communication with the LDAP server(s) can take place in two different ways, either insecure (the SIMPLE mechanism) or via an encrypted transmission (the SSL mechanism). Set the following variable accordingly:

```
TDBS_LDAP_MECHANISM={SIMPLE | SSL}
```

For example:

```
TDBS_LDAP_MECHANISM=SIMPLE
```

Provide here a maximum number of milliseconds that TRIP should wait for a response from the LDAP server(s).

```
TDBS_LDAP_TIMEOUT=1500
```

In order to find users, TRIP needs to be able to browse the LDAP repository. If the repository supports anonymous access for browsing, set the following variable to True, otherwise set it False.

```
TDBS_LDAP_ANONYMOUS={True | False}
```

For example:

```
TDBS_LDAP_ANONYMOUS=False
```

If anonymous browse access is not supported, you must provide the DN and credentials (password) for the user that will be used to perform browse operations when searching for users to authenticate. This is done using the following variables:

`TDBS_LDAP_USERNAME` is the fully qualified DN of the browse user

`TDBS_LDAP_PASSWORD` is the plain text of the browse user's password

The user specified must have read access to the entire tree descending from the root node provided by `TDBS_LDAP_BASE` (described below).

For example:

```
TDBS_LDAP_USERNAME=cn=Manager,dc=bjensen,dc=com
```

```
TDBS_LDAP_PASSWORD=thx1139
```

When attempting to authenticate a user, that user's identity will typically be provided as an RDN rather than a fully specified DN. In order to turn that RDN into a DN for authentication, you must provide the following set of variables:

`TDBS_LDAP_BASE` defines the base of the tree in which users can be found

`TDBS_LDAP_SEARCH` defines an LDAP search string to use to find users

For example, if the TRIP user community is collected in a subtree of the LDAP repository with a logical base of `ou=tox/o=pharma/c=us`, then the base of the search tree should be established as follows:

```
TDBS_LDAP_BASE=ou=tox,o=pharma,c=us
```

To find a user by RDN (for example by the UID or CN that the user presents as their typical login public key), specify an LDAP search string using the `%u%` substitution string to stand for the user's provided RDN. For example, when using the person structural schema (or some derivation, for example `organizationalPerson`, or `inetOrgPerson`) with the `uidObject` add-on schema the search string would be:

```
TDBS_LDAP_SEARCH=( &(objectclass=person) (uid=%u%) )
```

Any occurrence of the `"%u%"` pattern within the string will be replaced with whatever "username" is provided to TRIP during the login process.

Once the user has been found (i.e. their RDN has been dereferenced to a DN) their record must be turned into a TRIP username for use within the CONTROL database. The following variable is used to specify the field from the user record that will provide this mapping, for example in most user-related schemas, this would be the "uid" field:

```
TDBS_LDAP_MATCH=uid
```

Notes:

- (TRIPclassic or server based application)
- *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).

LDAPS

If using SSL for communication, the location of the local certificate database must be provided by setting the following variable. As TRIP uses the Mozilla LDAP SDK on Linux and Solaris, the database in question is that used by the Mozilla and Firefox browser applications (amongst others), is entitled "cert8.db" and can normally be found within a user profile, for example:

```
TDBS_LDAP_SSL_CERT_DB=/home/bjensen/.mozilla/cert8  
.db
```

On Windows TRIP uses the native LDAP SDK. For Windows installations of TRIP, the certificate database is the Windows certificate store of the local machine. In order for the SSL connection to work, the issuer of the SSL certificate in use by the LDAP server must be found in the Trusted Root Certification Authorities store.

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.

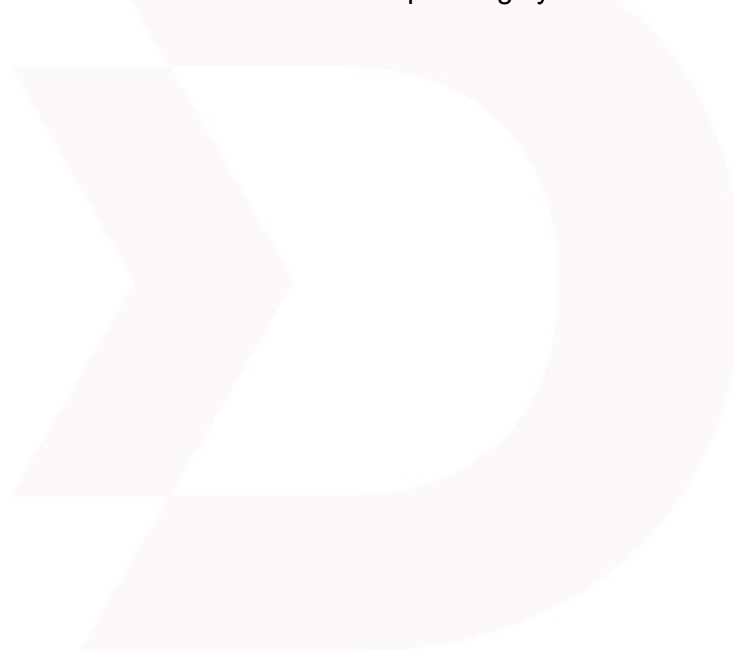
Notes:

- For instructions on how to enable local system validation, see the “Local system validation” section on page 250 of this guide.
- 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.



TRIP Grids

Notes:

- *TRIPgrids can only be administered using the TRIPmanager mmc snap-in and the information included here is for reference only. For further information, consult the TRIPmanager Administration Guide.*
- *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

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.

Classification Schemes

Notes:

- *TRIP Classification Schemes can only be administered using the TRIPmanager mmc snap-in and the information included here is for reference only. For further information, consult the TRIPmanager Administration Guide.*
- *The logical name **TDBS_Fel! Hittar inte referenskälla.** must be configured in the [Non-privileged] section your tdb.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

For more information on Classification Schemes, consult the TRIPmanager Administration Guide and the document

“TRIP_White_Paper_Classification.pdf”, available in the “doc” directory of the TRIP installation.

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 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 $^2F\text{text string}^{\wedge}$ and $^2F^1P\text{text string}^{\wedge}$ equivalent, and if field two in the current record is a new field, then both are equivalent to $^2F^1P\text{text string}^{\wedge}$. 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**, **Time** 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 **STatus** 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 **Status** 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 output format 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 aseaname[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 `tdbs.conf` configuration file.

To make the ASE library, use the following procedure:

1. Create a directory to hold your source files.
2. Copy all of the files from the ASE directory in the TRIP tree to your new directory.
3. Create your source files.
4. 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.
5. 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
MYASE2= c:\mylibs\myase2 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

UNIX

ASE routines written for UNIX can only be debugged through the use of multiple `printf()` statements. Other debugging methods, such as `dbx` et al, are not supported.

Windows

ASE routines written for Windows can only be debugged through the use of multiple `printf()` statements. Other debugging methods are not supported.

CCL ASEs

The CCL statement `CALl` invokes a named ASE routine with a user specified argument string, for example:

```
CALl 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 `CALl` 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:

- 0 no delay, call immediately
- 1 call when the user triggers a 'hot key' (normally <Gold><G> in TRIPclassic)
- 2 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 `Text` field and the number of part records in the record.

You can work around the first restriction, however, by defining a `Text` 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 STRING, 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:

1. 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.
2. 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 Tlme, 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.
7. 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 – TRIPclassic VT Numeric Keypad.....	10
Figure 1-2 The CONTROL database.....	18
Figure 1-3 Head and part records in a database.....	19
Figure 1-4 Carroll's head/part record structure.....	20
Figure 1-5 A head record.....	20
Figure 1-6 A part record.....	20
Figure 1-7 A record entity.....	21
Figure 1-8 A composite record.....	21
Figure 1-9 Record components.....	21
Figure 2-10 The Primary Option Menu.....	23
Figure 2-11 The Administration menu.....	23
Figure 2-12 The DB Design option.....	24
Figure 2-13 The Create/Modify Database option.....	24
Figure 2-14 The Create/Modify Database Design form.....	24
Figure 2-15 The General Database Properties form.....	24
Figure 2-16 The General Database Properties design keys.....	25
Figure 2-17 The Physical File fields.....	26
Figure 2-18 The Special Database fields.....	27
Figure 2-19 The Default Format fields.....	29
Figure 2-20 Sample SYSTEM default output.....	30
Figure 2-21 The Character Set fields.....	31
Figure 2-22 The Database Description field.....	34
Figure 2-23 The ASE overlay.....	35
Figure 2-24 The Sentence/Paragraph Separation overlay.....	35
Figure 2-25 The Sentence Separation fields.....	37
Figure 2-26 The Index/Update Submission overlay.....	42
Figure 2-27 The Field Definition form.....	44
Figure 2-28 The Field Definition form design keys.....	45
Figure 2-29 Naming and typing a field.....	45
Figure 2-30 The Field Definition form for a PHrase field.....	46
Figure 2-31 The Field Definition form for a TEXT field.....	46
Figure 2-32 The Field Definition form for a NUMber field.....	46
Figure 2-33 The Field Definition form for an INTEger field.....	46
Figure 2-34 The Field Definition form for a DATE field.....	47
Figure 2-35 The Field Definition form for a TIME field.....	47
Figure 2-36 The Field Definition form for a STRING field.....	47
Figure 2-37 The Field Attributes.....	48
Figure 2-38 Separate Indexing.....	50
Figure 2-39 The Field Attributes.....	51
Figure 2-40 The Field List overlay for Field Definition.....	58
Figure 2-41 The ASE overlay for Field Definition.....	58
Figure 2-42 The Accounting Information overlay.....	59
Figure 2-43 The Create/Modify Database option.....	61
Figure 2-44 The Create/Modify Database Design form.....	61
Figure 2-45 The Delete Database option.....	62
Figure 2-46 The Delete Database Design form.....	62
Figure 2-47 The Copy Database option.....	63
Figure 2-48 The Copy Database Design form.....	63
Figure 2-49 STATUS for database Alice, screen one of two.....	64

Figure 2-50	SStatus for database Alice, screen two of two.....	64
Figure 2-51	The Create/Modify Cluster option.....	66
Figure 2-52	The Create/Modify Database Cluster form	66
Figure 2-53	The Cluster Databases form	66
Figure 2-54	The Create/Modify Cluster option.....	67
Figure 2-55	The Create/Modify Cluster Design form	67
Figure 2-56	The Delete Cluster option	68
Figure 2-57	The Delete Database Cluster form.....	68
Figure 2-58	A SStatus screen for a cluster database.	69
Figure 3-59	The 'Train' thesaurus, vertical representation.....	71
Figure 3-60	The 'Train' thesaurus, horizontal representation	71
Figure 3-61	The Primary Option Menu	73
Figure 3-62	The Administration menu	74
Figure 3-63	The Thesaurus Design option	74
Figure 3-64	The Create/modify option.....	74
Figure 3-65	The Create/Modify Thesaurus Design form.....	74
Figure 3-66	The General Thesaurus Properties form	78
Figure 3-67	Naming and typing extra thesaurus fields	79
Figure 3-68	The field definition form for CTX.....	80
Figure 3-69	The field definition form for BTX.....	80
Figure 3-70	The field definition form for NTX.....	80
Figure 3-71	The field definition form for RTX.....	80
Figure 3-72	The field definition form for UFX.....	81
Figure 3-73	The field definition form for SNX	81
Figure 3-74	The field definition form for NRX	81
Figure 3-75	SStatus for thesaurus 'Thesali', screen one of two	83
Figure 3-76	SStatus for thesaurus 'Thesali', screen two of two	83
Figure 4-77	Sample accounting file.....	89
Figure 5-78	Data entry form design keypad functions	91
Figure 5-79	The Primary Option Menu	92
Figure 5-80	The Administration menu	92
Figure 5-81	The Data Entry Forms Maintenance menu.....	93
Figure 5-82	The Create/modify option.....	93
Figure 5-83	The Create/Modify Data Entry form.....	93
Figure 5-84	The Data Entry Form Definition screen	93
Figure 5-85	The Text Attributes menu.....	94
Figure 5-86	Text defined as 'Bold, Underline'.....	95
Figure 5-87	Bolded, underlined text	95
Figure 5-88	Entry box for 'Last Name'.....	95
Figure 5-89	The database field list	96
Figure 5-90	Field selection made	97
Figure 5-91	The default Field Data Options form.....	97
Figure 5-92	The Field Characteristics summary	98
Figure 5-93	The Field Attribute and Field Protect options.....	98
Figure 5-94	The Default values	99
Figure 5-95	The Default/Target Entry mode	99
Figure 5-96	The Recall Value of Preceding Record options	100
Figure 5-97	The Value Checking options	100
Figure 5-98	Tuple structure	101
Figure 5-99	Tuple Number	102
Figure 5-100	Other Switches	102
Figure 5-101	Coupled Entry Form.....	103
Figure 5-102	Link Status	104

Figure 5-103	The Link Definition form	104
Figure 5-104	Record-level ASEs for Data Entry form	106
Figure 5-105	Field-level ASEs for Data Entry form	106
Figure 5-106	Drawing the upper horizontal border	107
Figure 5-107	Drawing the perpendicular border	107
Figure 5-108	The finished border	107
Figure 5-109	The field ordering phase	108
Figure 6-110	Format layout and construction	110
Figure 6-111	Output format components	111
Figure 6-112	The Primary Option Menu	112
Figure 6-113	The Administration menu	112
Figure 6-114	The Output Format option	112
Figure 6-115	The Create/Modify option	113
Figure 6-116	The Create/Modify Output Format form	113
Figure 6-117	SStatus Carroll, screen one of three	138
Figure 6-118	SStatus Carroll, screen two of three	138
Figure 6-119	SStatus Carroll, screen three of three	138
Figure 6-120	Paged output	143
Figure 6-121	The Show Format window	146
Figure 7-122	The anatomy of a search form	210
Figure 7-123	The search form Alice_Demo2	211
Figure 7-124	Alice_Demo2's 'Change Format' command menu	211
Figure 7-125	Alice_Demo2's 'SHOW Menu' command menu	212
Figure 7-126	Alice_Demo2's 'PRINT Options' command menu	212
Figure 7-127	The Primary Option Menu	213
Figure 7-128	The Administration menu	214
Figure 7-129	The Forms/Procs Maintenance Menu	214
Figure 7-130	The Search Forms Maintenance Menu	214
Figure 7-131	The Create/Modify Search Form screen	214
Figure 7-132	Search Form Definition Screens, Page 1	215
Figure 7-133	Page 1 for Alice_Demo2	216
Figure 7-134	Search Form Definition Screens, Page 2	217
Figure 7-135	Sample search form layout	217
Figure 7-136	Sample layout tab order	218
Figure 7-137	Page 2 for Alice_Demo2	218
Figure 7-138	Search Form Definition Screens, Page 3	219
Figure 7-139	Page 3 of Alice_Demo2	223
Figure 7-140	Search Form Definition Screens, Page 4	223
Figure 7-141	The Next, Exception and Help defaults	225
Figure 7-142	Page 4 of Alice_Demo2	226
Figure 7-143	Search Form Definition Screens, Page 5	226
Figure 7-144	Page 5 of Alice_Demo2	227
Figure 7-145	Search Form Definition Screens, Page 6	227
Figure 9-146	The Primary Option Menu	240
Figure 9-147	The Administration menu	240
Figure 9-148	The Load/Index menu	241
Figure 9-149	The Index option	241
Figure 9-150	The Index New/Changed Database form	241
Figure 9-151	The Load/Index option	242
Figure 9-152	The Load and Index Database form	242
Figure 9-153	The Load option	243
Figure 9-154	The Load Database form	243
Figure 10-155	The Primary Option Menu	247

Figure 10–156	The Administration menu	248
Figure 10–157	The User option	248
Figure 10–158	The New User option	248
Figure 10–159	The Create New User form	248
Figure 10–160	The Delete User option	249
Figure 10–161	The Delete User form.....	249
Figure 10–162	The Profile option.....	249
Figure 10–163	The User Profile form.....	249
Figure 10–164	Changing the Date Format.....	250
Figure 10–165	The Date Form box	250
Figure 10–166	Changing the first date separator	250
Figure 10–167	Changing the second date separator	251
Figure 10–168	The Change UM option.....	251
Figure 10–169	The Transfer User Managership form	251
Figure 10–170	The Group option.....	252
Figure 10–171	The New Group option.....	252
Figure 10–172	The Create New Group form	252
Figure 10–173	The Delete Group option.....	252
Figure 10–174	The Delete Group form	252
Figure 10–175	The Add Member option.....	253
Figure 10–176	The Add Group Member form	253
Figure 10–177	The Delete Member option.....	253
Figure 10–178	The Delete Group Member form	253
Figure 10–179	The System option	254
Figure 10–180	The Manager Rights option	254
Figure 10–181	The Grant User/File Manager Rights form.....	254
Figure 10–182	The Password option	255
Figure 10–183	The Set Password form.....	255
Figure 11–184	The Primary Option Menu	256
Figure 11–185	The Administration menu	256
Figure 11–186	The DB Design option	257
Figure 11–187	The DB Access option.....	257
Figure 11–188	The Database Access Rights Definition form	257
Figure 11–189	Default access rights for user 'New'	258
Figure 11–190	Access to Carroll for user 'New'	259
Figure 11–191	The default Access Rights Definition overlay	260
Figure 11–192	Field Access Rights defined for user 'New'	260
Figure 11–193	Record-level access rights for user 'New'.....	261
Figure 11–194	New record-level access rights for user 'New'	261
Figure 11–195	The Change FM (File Manager) option	263
Figure 11–196	The Transfer File Managership form	263
Figure 11–197	Carroll's Show ACcess screen	264

Tables

Table 0–1	TRIP naming conventions	11
Table 1–2	Sample flat file table.....	13
Table 1–3	Sample relational database tables	14
Table 1–4	Sample full-text database table	15
Table 2–5	General database properties overlay forms.....	25
Table 2–6	Use of the record name field	28
Table 2–7	Special characters	31
Table 2–8	The character folding classes.....	32
Table 2–9	Truncation, masking and special symbols	33

Table 2–10	General database properties overlay forms.....	34
Table 2–11	The character classes	36
Table 2–12	Sentence definition in TRIP.....	39
Table 2–13	Paragraph definition in TRIP	41
Table 2–14	Field definition overlay forms.....	44
Table 2–15	Field attributes and restrictions.....	48
Table 2–16	Symbols used in pattern specification	52
Table 2–17	TRIP's predefined character sets	53
Table 2–18	A simple pattern	53
Table 2–19	A more complex pattern	54
Table 2–20	More patterns.....	55
Table 2–21	Sample combined character sets	56
Table 2–22	Modifying a database design.....	60
Table 3–23	Record contents and thesaurus design for 'Train'.....	72
Table 3–24	The thesaurus template.....	75
Table 3–25	Record contents and thesaurus design	76
Table 3–26	Hierarchical relationships of the 'Train' thesaurus	77
Table 4–27	Line types and the accounting log	87
Table 5–28	Default Entry Form Field Values	99
Table 5–29	Default Target Entry Modes.....	100
Table 6–30	Types of background text.....	124
Table 6–31	Text string reserved characters	124
Table 6–32	Headers	125
Table 6–33	Separators	125
Table 6–34	Trailers.....	128
Table 6–35	Text string functions.....	130
Table 6–36	Field type-dependent functions	132
Table 6–37	Box and box group functions.....	135
Table 6–38	Format functions	136
Table 6–39	FOR loop functions	137
Table 6–40	Structure of Olympic_Games	140
Table 6–41	Date formats	161
Table 6–42	Samples of <Numform> output.....	189
Table 7–43	Search form movement keys.....	212
Table 7–44	Examples of search form substitution rules	220
Table 7–45	Valid video attribute combinations.....	222
Table 8–46	Anatomy of a global update command	230
Table 8–47	Structure of a global update using record numbers	230
Table 8–48	Generic update targets.....	231
Table 8–49	Structure of a global update using a search result.....	234
Table 8–50	Record update targets.....	234
Table 9–51	Operating systems and log file names.....	243
Table 9–52	Running the BAFINI utility	244
Table 11–53	General field access rights	258
Table 11–54	Unsupported combinations of access rights	258
Table 11–55	General field access rights	259

Index

- symbol 33, 52, 247, 256
- ' symbol 32, 38
- ! symbol 33, 38, 124, 221, 281
- " symbol 32
- # symbol 33
- \$ symbol 33, 287
- %RTNA 292
- & symbol 33
- (symbol 38
- () symbol 33, 52
-) symbol 38
- * symbol 52, 96, 248, 255, 260
- . symbol 33, 38, 247, 281
- .. symbol 52
- / symbol 33, 52, 124, 247
- // symbol 52
- /symbol 38
- : symbol 39, 247
- ? symbol 33, 38, 281
- [symbol 38
-] symbol 38
- ^ symbol 278
- _ symbol 24, 124
- { symbol 38
- } symbol 38
- + symbol 33, 52
- < symbol 38, 110, 113, 124
- <_>, as convention 9
- <Append>** 137, 147
- <At_end>** 135, 148
- <Backspace>** 101, 108
- <Base>** 130, 149
- <Border>** 107
- <Call>** 286
 - format** 136, 150
 - text string** 130, 152
- <Case>** 135, 154
- <Chr>** 130, 156, 157
- <CR>** 36, 96
- <CR>**, as convention 9
- <CR><LF>** 280, 281
 - and TForm 279
- <Ctrl>**
 - <Ctrl><Y>** 22
- <Curdate>** 130, 158
- <Cut Field>** 105, 106
- <Dateform>** 130, 159
- <Debit>** 59, 136, 161
- <Delete Line>** 101
- <Delete><Field>** 61
- <Enter>** 9
- <Ff>** 130, 162
- <FF>** 36
- <FF>**, as convention 9
- <Field>** 58, 79, 96, 105
- <Field><Area>** 95
- <Fieldname>** 132
- <Fieldno>** 132
- <Fieldtype>** 132
- <FOR>** loops 164
- <Gold>** 91
 - <Gold><9>** 106
 - <Gold><Border>** 107
 - <Gold><C>** 105, 108
 - <Gold><Cancel>** 108
 - <Gold><D>** 57
 - <Gold><Delete Line>** 101
 - <Gold><Enter>** 44, 105
 - <Gold><Field Area>** 95, 106
 - <Gold><Field List>** 94
 - <Gold><Field>** 97
 - <Gold><G>** 293
 - <Gold><kp .>** 94, 97
 - <Gold><kp 6>** 106
 - <Gold><kp 9>** 94, 97, 248, 288, 304
 - <Gold><Leave>** 23, 24, 74, 253, 254, 306
 - <Gold><List>** 248
 - <Gold><N>** 105
 - <Gold><O>** 107
 - <Gold><Order>** 107
 - <Gold><PF4>** 106
 - <Gold><R>** 100, 108
 - <Gold><Select>** 94, 97, 102, 257
 - <Gold><T>** 103
 - <Gold><Tab>** 207
- <Gold>**, as convention 9
- <Hitlist>** 137, 167
- <Hits>** 130, 169
- <If-changed>** 135, 170
- <If-empty>** 135, 172
- <If-nonempty>** 135, 173
- <If-unchanged>** 135, 174
- <Indent>** 111, 135, 176
- <kp .>** 58, 94, 96, 97, 102, 108
- <kp ->** 107
- <kp 1>** 25, 34, 44, 58, 105, 106, 287, 304
- <kp 2>** 25, 34, 35
- <kp 3>** 25, 34, 42
- <kp 6>** 61, 105, 106
- <kp 7>** 25, 34, 44, 58, 59, 84
- <kp 8>** 95
- <kp 9>** 44, 58, 79, 96, 105

<kp>.....	34	◇ symbol.....	103
<kp>, as convention.....	9	Access	
<Leave>.....	23, 24, 44, 58, 74, 94, 97, 98, 108, 306	database	
<Leave>, as convention.....	9	cluster.....	259
<LF>.....	36, 39, 40, 50, 281	defining.....	255
<LF>, as convention.....	9	field level.....	256
<Link>.....	135, 178	first form.....	255
<Loop variables>.....	137	general field.....	255
<Next Page>.....	9, 34, 79, 97	hidden read scope.....	258
<Next>, as convention.....	9	hierarchy of access rights.....	258
<Next><Page>.....	44, 58	listing.....	261
<NL>, as convention.....	9	read.....	255, 256
<Noff>.....	135, 136, 181	read scope.....	255, 257, 258
<Nolf>.....	182	record level.....	257
<Noorig>.....	111, 135, 183	write.....	255, 256
<Numform>.....	130, 185	write scope.....	255, 257, 258
<Occs>.....	130, 187	print.....	261
<Once>.....	135, 188	show.....	261
<Orig>.....	135, 189	Access privileges	
<Page Down>.....	9	database.....	253
<Page Up>.....	9	Accounting log	
<Page>.....	59, 84	B-line.....	87
<Pageno>.....	130, 191	C-line.....	87
<Paragraphno>.....	132	E-line.....	87
<Parts>.....	130, 192	F-line.....	87
<Paste Field>.....	106	M-line.....	87
<PF1>.....	9	O-line.....	87
<PF3>.....	9, 304	Q-line.....	87
<PF4>.....	106	R-line.....	87
<Prev>, as convention.....	9	S-line.....	87
<Previous Page>.....	9, 98	U-line.....	87
<Return>.....	9	Added fields	
<Rid>.....	130, 193	thesaurus.....	79
<Ris>.....	130, 194	Adding	
<Rname>.....	130, 195	user group member.....	250
<Select>.....	58, 79, 94, 96, 97, 102, 108	Administrator	
<Sentenceno>.....	132	database.....	243
<Sortfields>.....	111, 136, 196	database, granting privileges.....	251
<SP>		Alice database.....	8, 9, 30
and TForm.....	279	Status.....	64, 83
<Subfieldno>.....	132	Append	
<Subrid>.....	130, 197	data entry form.....	99, 100
<Substring>.....	130, 198	Application Software Exit.....	see ASE
<Tab>.....	50, 95, 97, 101, 102, 103, 108, 281, 308	Arglen.....	289, 292, 293, 296, 300, 301
<Text variables>.....	136, 200	Argstr.....	289, 292, 293, 296, 300, 301
<Timeform>.....	130, 202	ASE	
<Trace>.....	135, 201	baffit_	
<Undelete Line>.....	101	ase.....	286
<VT>.....	36	ase1.....	286
<VT>, as convention.....	9	ase2.....	286
<Weight>.....	130, 204	box-based.....	304
> symbol.....	38, 110, 113, 124	entry.....	307
		exit.....	307

box-based (TRIPclassic only)	307	BAF	21, 226, 236
CCL	286, 292	and general database properties	25
data entry		and LOAD procedure.....	278
field level	288	and TForm.....	284
record level.....	287, 288	and the record name field	28
data entry (TRIPclassic only)	287, 304	BAFFINI.....	241
database design	34	BAFFIT	295, 297, 298, 299
debugging.....	291	MODE	297, 298, 299
directory.....	290	Baffit_	
entry form	105, 106	ase	286, 295
FAIL.....	292	ase1	286, 299
field-specific.....	295	ase2	286, 299
structured	296	Base access	
unstructured	297	print.....	261
form initialization (TRIPclassic only).....	305	show.....	261
format	289	Base file.....	21
format-level.....	294	Base File	see also BAF
form-based	304	Base index file	21, 22
form-based (TRIPclassic only).....	305	Base Index File.....	see also BIF
index.....	287, 300	Base_spec_rec.....	299
library.....	289, 290	BaseSpecRecord.....	299
linking to TRIP	289	Batch	
quitting form with <Leave> (TRIPclassic		update	8
only)	306	BIF.....	21, 22, 226
record commit		and general database properties	25
after writing to BAF	306, 307	and the record name field	28
record-specific	299	Bigram.....	22
reports	286, 292	B-line	87
RET CODE	292	Bold, as convention	9
scanit_ase	287	Border	
search form.....	218	data entry form	107
search form (TRIPclassic only)	288, 308	Box.....	111
template, in C	289	constituents	111
text insert.....	293	definition.....	111
TForm load	286, 295	functions.....	134
TRIPclassic callback functions.....	308	group	
TRIPkernel callback functions.....	308	<i>definition enclosures</i>	122
uses of.....	286	output format.....	122
ASE_		layout	
CONTINUE.....	305, 308	defining a	113
FAIL.....	289, 301, 302	numbering	116
FIXFIELD.....	289, 305	output format	
MESSAGE.....	305	header.....	144
NOFIELD	305	trailer.....	144
REFRESH	289, 292, 305	page level.....	143
SUCCESS	289, 292, 301, 302	positioning	116
ASEOBJ	290	using coordinates	116
Attributes		using preceding boxes	117
search form.....	218	proportioning	
Auto tab	102	using columns	121
Background text		using lines	120
entry form	92	using lines and columns	119
output format	123	search form	

command	205	Character masks, as searchable characters	33
search result.....	205	Character sets	31
simple	113	specification	52
size		Characters	
output format	119	<i>control skip</i>	279
<i>specifications</i>		reserved	124
<i>directed</i>	116	searchable.....	32
nonspecific	116	searchable special.....	31
Box/box group		special	31
functions		CHECK ENTRY	305
<append>	147	Chevron, as convention	9
<at_end>	135, 148	C-line	87
<case>	135, 154	Columnar output	145
<if-changed>	135, 170	Command	
<if-empty>	135, 172	box	
<if-nonempty>	135, 173	search form	205
<if-unchanged>	135, 174	DEfine	9
<indent>	135, 176	EForm	30
<link>	135, 178	Format	30
<noorig>	135, 183	INDex	238
<once>	135, 188	search form	
<orig>	135, 189	default	
<trace>	135, 201	exception	222
CALI.....	286, 292	help	222
Carriage return		next.....	222
output format	124	exception.....	222
Carroll database	8	help	222
<i>chapter information in</i>	19	id221	
<i>page information in</i>	20	menu size.....	212
records		next.....	221
chapter	20	specifications	220
main	20	search form menu window.....	206
paragraph.....	20	search forms	220
part.....	20	variables.....	223
Status	283	Component	
Case sensitivity		in head/part database.....	21
and global updating	235	<i>Composite record</i>	19, 21
CCL		Constituent	
<i>and</i>		box	111
<i>TRIPclassic</i>	10	CONTROL database	17, 244, 245
commands and output formats.....	146	and search form errors	224
search menu option	9	contents.....	18
CCLASE.C	292	Control master, and TForm.....	279
Changing pages		<i>Control skip characters</i>	279
data entry form	106	<i>Control strings</i>	
Character folding class		<i>and TForm</i>	278
and diacritics.....	32	Conventions	
and umlauts.....	32	<_>	9
default.....	31	<CR>	9
ENGLISH	32	<FF>	9
MULTinational.....	31	<Gold>	9
specification.....	31	<kp>	9
SWEdish.....	32	<Leave>	9

<LF>	9	deleting	109
<Next>	9	drawing borders	107
<NL>	9	field attributes	98
<Prev>	9	insert	99, 100
<VT>	9	moving entry field	106
boldface	9	replace	99, 100
chevrons	9	size	93
Courier fonts	9	tuple	101
italic	9	creating	102
lower case	9	navigating	101
naming	11	Database	
space character	9	access	
upper case	9	cluster	259
Copying		defining	255
data entry form	108	field level	256
output formats	145	first form	255
records with TForm	285	general field	255
with global updating	234	hidden read scope	258
Corr database	8	hierarchy of access rights	258
and output formats	113	listing	261
<i>database field numbers</i>	281	privileges	253
STatus	138	read	255, 256
structure	281	read scope	255, 257, 258
Courier fonts, as convention	9	record level	257
Creating		write	255, 256
data entry form	92	write scope	255, 257, 258
output formats	112	administration	8
search forms	210	administrator	18, 23
user	244	Alice	8, 30
user group	248	STatus	64, 83
Current		Carroll	8
command, search form	207	chapter information in	19
date form	247	paragraph information in	20
CURRENT ITEM	295, 298, 299, 305, 307	records	
Cut	28	chapter	20
Cutting and pasting		main	20
data entry form	106	paragraph	20
Data		part	20
models	13	STatus	283
normalization	14	cluster	
organization, and TRIP	15	output formats and	145
shown but not stored	102	CONTROL	17, 244, 245
Data entry		contents	18
ASE	105, 106	Corr	8
entry fields	92	<i>database field numbers</i>	281
entry form	91	description	
append	99, 100	and general database properties	34
changing pages	106	and STatus	34
copying	108	design	24
creating	92	saving	59
cutting and pasting	106	<i>field numbers</i>	281
default	29	general properties	24
defining field entry area	95	head/part	

component	21	output formats	145
head record	20	records with TForm	284
part record	20	user	245
record	21	user group	249
record entity	20	user group member	250
management system		Deletion marker	284
full-text	13, 14	Delimiter	
reindexing	241	TForm	278
relational	13, 14	Delineators, as searchable characters	33
responsibility, transferring	260	Design	
security	8	database	24
Thesali	8	Device/directory	
TRIP basics	19	database	26
Database administrator	243	Digits, as searchable characters	32
granting DA privileges	251	Directory	
Database Administrator..see also File Manager		ASE	290
Database Corr		INCLUDE	289
structure	281	SAMPLES	292, 294, 296, 303
Databases		Document, and TRIP	16
listing	64	Dump output format	132
Date		Element	
current	99	output format	111, 123
form, current	247	E-line	87
DAtE	17	ENGLISH	
restrictions	56	character folding	32
DEBIT.LOG	59, 84, 85	Entity	
Default		record	20
data entry form	29, 92	Entry fields	
design-time, in data entry forms	99	data entry	92
help		data entry, moving	106
search form	219	<i>Entry form</i>	
output format	29, 30	background texts	92
run-time, in data entry forms	99	creating	92
Define		<i>data entry</i>	91
field entry area	95	append	99, 100
space character	33	changing pages	106
DEfine	9	copying	108
EForm	30	cutting and pasting	106
Format	30	defining field entry area	95
Definition		deleting	109
box	111	drawing borders	107
Delete	226	field attributes	98
field	226	insert	99, 100
paragraph	226	replace	99, 100
record	226	size	93
sentence	226	date, current	99
string	226	default	92
subfield	226	link database	104
DELeTe	227, 228	link status	104
DELETE ITEM	298	listing	109
DELETE RECORD	307	OS user, current	99
Deleting		source field	104
data entry form	109	time, current	99

TStamp, current.....	99	File	
user, current	99	flat	13
ERRLOG	240	<i>inverted, and TRIP</i>	21
Error checking		structures	
global updating	236	in TRIP	21
Exit		File manager	23, 243
application software	105, 106	granting FM privileges	251
F marker	278, 280	FIXFIELD.....	306, 308
Field		Flat file.....	13
attributes, in data entry forms.....	98	F-line	87
data options	97	FOR loop	
delete.....	226	and output formats	136
elements		functions	164
and output formats.....	115	<append>	137
entry area, defining	95	<hitlist>	137, 167
head	19	<loop variables>	137
insert.....	226	Form	
list.....	58	<i>data entry</i>	91
marker		errors	
TForm.....	278, 280	and search forms	224
part.....	19	message line, search form.....	206
record name	28	overlay	25
record number	28	report line, search form.....	206
record part name	28	Format	
replace.....	226	functions	
search form.....	205	<call>	136, 150
selecting	96	<debit>	136, 161
source.....	104	<noff>	136, 181
types		<nolf>	182
DAte	17	<sortfields>	136, 196
in TRIP	16	<text variables>	136, 200
NUmber.....	17	output	110
PHrase	16	Forms	8
STring.....	17	<i>Fragment index</i>	
TExt.....	16	<i>and TRIP</i>	14
Tlme	17	Free text	15
<i>Field numbers</i>	281	Full-text database management system	
Field_spec_rec	295, 300	(TDBS)	13, 14
Fields		Function keys	
added		search form	209
thesaurus	79	Functions	
data entry	92	output format	129, 135
hidden.....	259	box	134
and data entry	260	text string	129
and output formats.....	259	<i>FUZZ, and the VIF</i>	22
and searching.....	259	G marker	278, 280
long.....	96, 103	General database properties	
read-protected	259	character sets.....	31
and data entry	260	database description	34
and output formats.....	259	default	
and searching.....	259	data entry form	29
sticky	100	output format.....	29
FieldSpecRecord	295, 300	design keys for	25

overlay forms	34	record	19, 20
physical files	25	Head/part database	
special database fields	27	component	21
transaction log	26	head record	20
General search form properties.....	212	part record	20
General Settings, Limits and Defaults	263	record	21
CCL Command Length Limit.....	263	record entity	20
Chinese GBK Character Set.....	263	Header	259
Database File size Limit in UNIX.....	264	output format	125
DEfine command defaults.....	264, 265	Header_Box	
Euro Currency Symbol		output format	144
Character Set	263	Help	
Searching for	263	database, search form.....	213
Open databases limit	264	search form	
GET LINE	305, 308	default	219
Global updating.....	226	Hidden fields.....	259
and case sensitivity.....	235	and data entry	260
and log file	236	and output formats	259
copying with.....	234	and searching.....	259
DElete	227, 228, 229	INCLUDE directory	289
error checking.....	236	Indent	
examples		reports	176
using DElete.....	230, 233	Index	
using INSert.....	229, 232	files, in TRIP	21
using UPDate	230, 232	fragment	14
INSert	227, 228, 229	INDEX	
part records	234	command	238
records	230	Indexing.....	237
targets	228	Insert	28, 226
type.....	227	data entry form	99, 100
update		field	226
domain	228	paragraph.....	226
target.....	228, 231	sentence.....	226
type	227, 231	subfield.....	226
value.....	228, 231	INSert	227, 228
UPDate.....	227, 228, 229	INteger	
upper- and lower-case letters.....	235	restrictions	56
using a search result.....	230	Interval	
using record numbers	227	and values list.....	56
Group		in PHrase pattern	52
PUBLIC	248	Italic, as convention	9
user	243, 258	IX databasename unique ID.log.....	240
creating	248	Keep all sets	
deleting.....	249	search form	212
Group member		Keypad Emulation	
user		< - >.....	266
adding	250	< , >.....	266
deleting.....	250	< . >.....	266
Groups		< Backspace >.....	266
listing	252	< Page Down >.....	266
Hashed tables.....	21	< Page Up >	266
Head		<PF1>	266
field.....	19	<PF2>	266

<PF3>.....	266	user, granting privileges	251
<PF4>.....	266	Marker	
Key combinations	266	deletion.....	284
keypad < Enter >	266	field	
Keys		TForm	280
for general database properties design	25	<i>length</i>	285
Layout box		record	
defining a.....	113	name.....	284
Layout retained		name, TForm	280
and TForm	279, 281	part, TForm	280
LD databasename unique ID.log	240	TForm	280
Legal values		sentence, TForm	280
search forms	224	subfield, TForm	280
<i>Length</i>		zero	
<i>marker</i>	285	field	284
Letters, as searchable characters	32	record.....	284
LI databasename unique ID.log	240	<i>Maximum number of boxes</i>	
Linefeed		<i>search form</i>	215
output format	124	MESSAGE.....	299, 306
Link.....	104	Meta-record	16, 19, 21
database, entry form	104	M-line	87
status, entry form	104	MULTinational	
List		character folding.....	31
<i>values and intervals</i>	56	N marker.....	280
Listing		Name	
databases.....	64	search form	212
entry forms.....	109	<i>user</i>	245
groups	252	Naming conventions	11
Literal		database	24
search form.....	205	Natural language text.....	15
LOAD.....	278	<i>NRX field</i>	75
Log file		NUmber	17
and general database properties	25	restrictions	56
naming.....	240	Numbering	
<i>Logical names</i>		box	116
<i>device/directory</i>	26	O-line.....	87
Logical Names		Olympic_Games database.....	140
SUPERMAN	243	structure	140
Long fields	96, 103	Operators, as searchable characters	33
Lower case, as convention.....	9	OS User	
Main record.....	19	current.....	99
TForm.....	283	<i>Output</i>	
Manager		<i>paged</i>	143
file.....	243, see Database Administrator	Output format.....	110
file, granting privileges	251	and database clusters	145
privileges		and FOR loops	136
granting	250	background text.....	123
in TRIP	18	box	
privileges, removing.....	251	functions.....	134
responsibility, transferring	260	group.....	122
system	18, 23, 243	size	119
privileges	250	carriage return	124
user	18, 243, 244	copying.....	145

creating.....	112	reserved characters in	124
default.....	30	sample	132
deleting.....	145	separator	125
dump	132	series of spaces or tabs.....	124
element.....	111, 123	<i>specification file</i>	113
functions	129, 135	text functions	
<append>	137	field-dependent	131
<base>	130	text inserts.....	129
<call>--format	136	text strings	
<call>--text string	130	field-dependent	125
<case>	135	field-independent.....	129
<chr>	130	<i>timestamp</i>	115
<curdate>	130	trailer	128
<dateform>	130	trailer_box	144
<debit>	136	<i>TStamp</i>	115
<ff>	130	Output Format Reference Guide.....	147
<fieldname>	132	Output formats	
<fieldno>	132	and database Corr.....	113
<fieldtype>	132	and specific field elements.....	115
<hitlist>	137	Output in columns	
<hits>	130	output format	145
<if-changed>	135	Overlay form, and database creation	25
<if-empty>	135	P marker.....	278, 280
<if-nonempty>.....	135	Page	
<if-unchanged>.....	135	control	143
<indent>	135	level	
<link>	135	box.....	143
<loop variables>	137	<i>Paged output</i>	143
<noff>	135, 136	Paragraph	
<noorig>	135	delete	226
<numform>.....	130	insert	226
<occs>	130	marker	
<once>	135	TForm	280
<orig>	135	replace	226
<pageno>	130	<i>Part</i>	
<paragraphno>	132	field	19
<parts>	130	<i>record</i>	19, 20
<rid>	130	Part record	
<ris>	130	global updating	234
<rname>	130	Paste	28
<sentenceno>	132	Pattern	
<sortfields>	136	PHrase field.....	52
<subfieldno>	132	PHrase	16
<subrid>	130	PHrase field	
<substring>.....	130	pattern.....	52
<text variables>	136	<i>pattern interval</i>	52
<timeform>	130	Positioning	
<trace>	135	box	116
<weight>	130	using coordinates	116
header	125	using preceding boxes	117
header_box.....	144	Print	
linefeed.....	124	access.....	261
output in columns.....	145	base access	261

group	252	deleting	
user	252	with TForm	284
Privileges		global updating	230
database access	253	in TRIP	19
database administrator, granting	251	part	
file manager, granting	251	global updating	234
first access form	255	replacing with TForm	284
manager		updating with TForm	284
removing	251	Reindexing	241
manager, granting	250	Relational database management system	
user manager, granting	251	(RDMS)	13, 14
Profile		Replace	226
user	246	data entry form	99, 100
Proportioning		field	226
box		paragraph	226
using columns	121	sentence	226
using lines	120	string	226
using lines and columns	119	subfield	226
PUBLIC	248	Replacing	
PUT LINE	305, 308	records with TForm	284
Q-line	87	Reports	
R marker	278, 280	functions	
Read protection	259	<append>	147
and data entry	260	<at_end>	135, 148
and output formats	259	<base>	149
and searching	259	<call>-format	150
Record		<call>-text string	152
<i>composite</i>	19, 21	<case>	154
delete	226	<chr>	156, 157
description of a		<curdate>	158
in head/part database	21	<dateform>	159
entity	20	<debit>	161
head	19, 20	<ff>	162
main	19	<FOR> loops	164
marker		<hitlist>	167
TForm	280	<hits>	169
meta-	16, 19, 21	<if-changed>	170
name	28	<if-empty>	172
name field	28	<if-nonempty>	173
name marker	284	<if-unchanged>	174
name marker, TForm	280	<indent>	176
name value	28	<link>	178
number field	28	<noff>	181
<i>part</i>	19, 20	<nolf>	182
marker, TForm	280	<noorig>	183
part name field	28	<numform>	185
part, TForm	283	<occs>	187
unit, in head/part database	21	<once>	188
user	245	<orig>	189
Records		<pageno>	191
copying		<parts>	192
with global updating	234	<rid>	193
with TForm	285	<ris>	194

<name>	195	specification	220
<sortfields>	196	variables	223
<subrid>	197	creating	210
<substring>	198	current command	207
<text variables>	200	elements	205
<timeform>	202	command boxes	205
<trace>	201	literals	205
<weight>	204	result boxes	205
indent	176	search fields	205
Reserved characters	124	errors and CONTROL	224
! symbol	124	form	
/ symbol	124	message line	206
_ symbol	124	report line	206
< symbol	124	function keys	209
> symbol	124	general properties	212
Restrictions		help	
Date	56	database	213
in search forms	218	default	219
INteger	56	keep all sets	212
NUmber	56	legal values	224
<i>Time</i>	57	<i>maximum number of boxes</i>	215
to valid values	51	name	212
R-line	87	restrictions	218
S marker	278, 280	search box	209
SAMPLES directory	292, 294, 296, 303	search entry window	206
SCANASE.C	303	start	
Scanit_ase	287, 300	command	213
Search box		procedure	212
search form	205, 209	start procedure	212
specification/search logic	216	TRIP message line	206
Search entry window		type	212
search form	206	variable value description	224
Search form		Search results	
anatomy	206	boxes for	205
command window	207	Searchable characters	32
form message line	208	and ' symbol	32
form report line	207	and ! symbol	33
form result box	207	and " symbol	32
search entry window	207	and # symbol	33
TRIP message line	208	and \$ symbol	33
ASE	218	and & symbol	33
attributes	218	and () symbol	33
command	220	and . symbol	33
default		and /	
exception	222	symbol	33
help	222	and ? symbol	33
next	222	and + symbol	33
exception	222	and character masks	33
help	222	and delineators	33
id 221		and operators	33
menu size	212	and sentence separator defaults	33
menu window	206	and space character	33
next	221, 222	and truncation symbols	33

and word masks.....	33	STring.....	17
Security		Structured query language.....	14
database.....	8	Subfield	
Select.....	28	delete	226
Sentence		insert	226
Delete	226	marker	
insert.....	226	TForm	280
marker		replace	226
TForm.....	280	SUPERMAN	243
replace.....	226	SWEdish	
separator	281	character folding.....	32
separator defaults, as searchable characters		symbol	9, 38
.....	33	System	
Separator		TRIP basics.....	13
output format	125	SYSTEM.....	88, 243, 246, 251, 260
sentence	281	System manager	18, 23, 243
Series of spaces or tabs		privileges	250
output format	124	Tab	
Session index file	22	as convention	9
SET ENTRY.....	305, 306, 308	order.....	102, 107
Show		Table	
access	261	hashed	21
base access.....	261	tdbs.conf.....	290, 291
group	252	TDBS_	
user	252	ACCDIR	84
SIF.....	22, 85, 88	ACCFLG.....	84, 85, 88
Simple box	113	ASELIBS	290, 291
Size		BATCH.....	237
data entry form	93	ERRMAILST.....	241
S-line	87	EXE/bafini	241
Source field		LOG	240
entry form	104	SIF	85
Space character		SYS.....	84
as convention	9	Text	
defining.....	33	attributes in data entry forms	94
Special characters		fixed screen in data entry forms.....	94
searchable	31	free.....	15
Specification file		natural language.....	15
output format	113	TEst.....	16
Specification/search logic		Text functions	
search box.....	216	output format	
SQL	14	field-dependent	131
Start		Text inserts.....	259
command, search form	213	output format	129
module, user profile	247	Text string	
procedure, search form.....	212	definition of, in TForm.....	279
procedure, user profile	247	functions	
STatus		<base>	130, 149
and database Alice	64, 83	<call>.....	130, 152
Sticky fields.....	100	<chr>	130, 156, 157
String		<curdate>.....	130, 158
delete.....	226	<dateform>	130, 159
replace.....	226	<ff>	130, 162

<fieldname>	132	R marker	278, 280
<fieldno>	132	record marker	278, 280
<fieldtype>	132	record name marker	280
<hits>	130, 169	record part	283
<noff>	135	record part marker	278, 280
<numform>	130, 185	replacing records with	284
<occs>	130, 187	S marker	278, 280
<pageno>	130, 191	sample file	282
<paragraphno>	132	sentence marker	278, 279, 280
<parts>	130, 192	subfield marker	280
<rid>	130, 193	text string, definition of	279
<ris>	130, 194	updating records with	284
<rname>	130, 195	zero field marker	284
<sentenceno>	132	zero record marker	284
<subfieldno>	132	Thesali database	8
<subrid>	130, 197	Thesaurus	
<substring>	130, 198	added fields	79
<timeform>	130, 202	top terms	73
<weight>	130, 204	Time	
output format	129	current	99
Text strings		Time	17
and ! symbol	124	restrictions	57
and / symbol	124	Timestamp	
and _ symbol	124	output format	115
and < symbol	124	Top terms	
and > symbol	124	thesaurus	73
and TForm	279	Trailer	
output format		output format	128
field-dependent	125	Trailer_Box	
field-independent	129	output format	144
TEXTASE.C	294	Trigram	22
TFOFIELD.C	296	TRIP	
TForm		and inverted file organization	21
and <CR><LF>	279, 280, 281	database basics	19
and <SP>	279	index files in	21
and control master	279	jump table	290
and control strings	278	manager privileges	18
and layout retained	279, 281	naming conventions	11
and text strings	279	records in	19
and the BAF	284	search form message line	206
copying records with	285	system basics	13
deleting records with	284	system data dictionary	see CONTROL
deletion marker	284	Truncation symbols, as searchable characters	33
F marker	278, 280	TStamp	
field marker	278, 280	current	99
G marker	278, 280	output format	115
log file	26	Tuple	101
main record	283	creating	102
N marker	280	navigating	101
P marker	278, 280	Type	
paragraph marker	280	search form	212
paragraph/subfield marker	278, 279	U-line	87
paragraphs and sentences in	281		

Unigram	22	individual	243, 244
UPDate	227, 228	<i>name</i>	245
Updating		<i>password</i>	245
global	226	print	252
and case sensitivity	235	print user group	252
and log file	236	record	245
copying with	234	responsibility, transferring	248
DElete	227, 228, 229	show	252
error checking	236	show user group	252
INSert	227, 228, 229	User group	243
part records	234	User manager	18, 243, 244
targets	228	granting UM privileges	251
type	227	User profile	246
update		date form separator characters	247
domain	228	start module	247
target	231	start procedure	247
type	231	Validation options	
value	228, 231	flag first occurrence	101
UPDate	227, 228, 229	immediate checking	100
using a search result	230	in data entry forms	100
using record numbers	227	<i>Values and intervals list</i>	56
Upper case, as convention	9	Variable value description	
User		search form	224
administration	243	VIF	21, 22, 226
creating	244	and general database properties	25
current	99	Vocabulary index file	21, 22
deleting a	245	Vocabulary Index File	see also VIF
end	244	Word	
group	243, 258	masks, as searchable characters	33
creating	248	WRITE MESSAGE	305
deleting	249	Zero	
member, adding	250	field marker	284
member, deleting	250	record marker	284