Relate

Data Maintenance for NonStop Servers

User Manual

Ascert

taking systems to the edge

For further information:

Ascert, LLC
759 Bridgeway
Sausalito, CA 94965

Telephone: (415) 339-8500
Email: info@ascert.com
Web: http://www.ascert.com

Table of Contents

Chapter One

INTRODUCTION

Welcome

Welcome to the RELATE family of products. If you need assistance with the development and maintenance of databases and database access programs within the Guardian processing environment of Compaq's Himalaya range of computer systems, then we are sure that you will gain benefit from these products.

RELATE provides intelligent manipulation and comparison functions on all Enscribe files within Compaq's Himalaya systems. By accessing the definition of your files within your DDL dictionary or COBOL copybooks, the RELATE products can:

- Compare sets of records from differing databases down to the field level, using a number of strategies, even where the layouts may be different.
    - Display the contents of nominated fields within selected records, intelligently applying group and field redefinitions.
    - Globally modify individual fields of selected records within your database.
    - Globally delete selected records from the database.
    - Insert new records into the database.
- Copy selected records between databases, converting fields of different formats.

The Relate Family of Products

The RELATE family consists of the following products:

- The base product, RELATE, is the core product upon which all other family members are built. It enables the creation, viewing, updating, comparison and deletion of records at the field-level.
- RELATE+ provides more advanced capabilities to assist with the processing of nonstandard DDL records and fields, and the migration / conversion of data from one database to another. These features make RELATE+ well suited for processing data produced by front-end communications processes, and for use in database migration projects.
- RELATE24, developed in co-operation with ACI Worldwide, is similar to RELATE+ and is optimized to process data files from ACI's Base24 product.
- RELATE/SQL is an external module accessed through RELATE that provides the ability to compare SQL tables.

The name RELATE will be used throughout the rest of this manual to refer generally to functionality present in all products of the RELATE family. Where functionality is specific to a particular family member, then this will be detailed when the functionality is described.

How this document is organized

We want you to be as productive as possible, as soon as possible. To help you determine how to use this manual, here is a quick outline of the contents:

Chapter 1	- Introduces the document
Chapter 2	- Provides a "quick-start" guide to using RELATE. It covers basic commands, and gives a quick tour of the product in operation. The chapter can be used as a tutorial.
Chapter 3	- Provides a user guide, discussing some of the concepts that RELATE uses, and describes the various areas of operation.
Chapter 4	- Gives a complete reference to all of the Relate Commands.
Appendix A	- Details how to install RELATE on your system, and lists the system management issues surrounding the installation
Appendix B	- Details the run-time parameters that RELATE will accept at startup to modify aspects of its internal operation.
Appendix C	- Details some of the common user messages generated by RELATE, together with their causes and solutions.
Appendix D	- Describes how to customize your copy of RELATE.
Appendix E	- Details the known limitations of using RELATE on Compaq Himalaya platforms.

Syntax

The syntax used throughout this manual for describing language statements is described below:

UPPERCASE LETTERS	Uppercase letters represent keywords; these can be entered exactly as shown, or can be abbreviated as long as the abbreviation unambiguously identifies the keyword.
<lowercase letters>	Lowercase letters within angled brackets represent variable entries that the user supplies.
Brackets []	Brackets enclose optional syntax items.
Braces {}	Braces enclose required syntax items; only one may be chosen from lists that are vertically aligned.
Ellipsis ...	An ellipsis following a pair of brackets or braces indicates that the enclosed syntax can be repeated zero or more times. Note there is a shorthand method for defining comma separated items supplied by the user. Longhand this would be as follows: <group> [, <group>] ... This can be shortened to the following: <group> [, ...]
Quotation Marks '	These are sometimes used to distinguish characters that are part of the language specification, from meta-characters; the quotation marks should not be entered by the user. For example, in the following:

	<field> '=' <value> The equals sign is surrounded in quotation marks, indicating that it forms part of the syntax definition. It can be entered (without quotes) by the user.

Chapter Two

QUICK START

Introduction

This chapter provides a quick introduction to using RELATE. New users should be able to read this and immediately feel comfortable with the basic operation of the product.

This chapter can be used as a tutorial: you will be able to perform the examples yourself, assuming that the product has already been installed on your system. If this is not the case, refer to Appendix A for details on how to accomplish this.

In order to try out the examples you will have to make your TACL's current default volume / subvolume that of RELATE's Installation Subvolume (this is normally REL8ISV). Additionally the example files produced during product installation must still be present in the subvolume, and you must have read / write file access to them.

Getting Started

RELATE is a standalone server-based program, run directly from the TACL command line. It provides intelligent manipulation and comparison of Enscribe files in Compaq’s Himalaya-based Guardian environment.

RELATE uses a conversational command interface, using syntax with which users familiar with Guardian subsystems will feel immediately at home. Users can enter RELATE commands on the startup command line, interactively in response to RELATE prompts, or programmatically within a command file; this last method facilitating the integration of RELATE into users' own environments.

In order to use RELATE, the operator first OPENs a file they wish to work on - the *current* file. They can then immediately perform various actions on either the entire opened file, or a subset of records in it - this latter method of operation is achieved by qualifying the actions with a WHERE clause, similar to Enform / SQL inquiries. Available actions include the following:

- List record contents.
    - Update / delete existing records.
    - Copy existing records between files.
    - Insert new records
- Compare records from different files

An example of this follows; you can try these commands out yourself (note that the user input has been underlined):

```
TACL 1 > RUN RELATE
RELATE (Rev 2.6.1 - 17-JAN-2007)
Relate> HELP
```

```
The following commands are available:

    CLEAR                   COMPARE                 COPY
    COUNT                   DELETE                  EXIT
    FC                      HELP                    HISTORY
    INFO                    INSERT                  LIST
    LOG                     OBEY                    OPEN
    OUT                     SET                     SHOW
    UPDATE                  VOLUME                  X
    XSQL

Relate> HELP OPEN
Syntax:  OPEN [<file> [<open-options>]]

  Note:  Opens the Enscribe file specified, and makes it the current file.  If
         no file is specified, then the name of the file currently open will be
         displayed, together with its open mode.

         By default, when a file is opened it will be opened for shared
         read-only access.  This can be overridden using the <open-options>.

Relate> HELP <OPEN-OPTIONS>
Syntax:  <open-options>  ::=   { FORUPDATE  }
                               { FORXUPDATE }

Relate> OPEN EGDDL
Relate> OPEN
Current file: $DATA1.REL8ISV.EGDDL (Read-only)
Relate> LIST WHERE $LINENUM < 5
    1.   : * This DDL source file is used to generate the dictionary
    2.   : * and files used in the examples contained within the
    3.   : * RELATE User Guide
    4.   : *
    5.   : ?DICTN !, COMMENTS
5 Records listed
Relate> OPEN EGFKS FORUPDATE
Relate> SET PRESENTATION HEX

PKEY:...   39 32 30 30 30 31 63 00                          | 920001c.

PKEY:...   39 32 30 30 30 32 63 00                          | 920002c.
Delete (Y/N/A/Q)? A

Relate> EXIT
```

In this example, the user first requested some help information. Note that the list of commands displayed includes the standard set of Guardian subsystem commands (for example OPEN, SHOW, FC, EXIT, etc.). Following display of more detailed help text regarding the OPEN command, the edit-type file EGDDL was opened for read-only access, and the first 5 lines LISTed. The default display options were in effect, therefore all of the first five lines of the file were displayed in ASCII. An alternative way for the user to display a specific number of records is achieved using the SET LISTING COUNT command; this limits the number of records displayed at any one time, and prompts the user for instructions once the records are displayed.

A new key-sequenced file EGFKS was then opened for update access. The user wanted to interactively delete all of the records in the file. Since they would be prompted as to whether to delete the individual records by (in this default case) displaying the primary key fields, they set the default method of presentation to be HEX to allow binary data to be displayed.

The user then executed the DELETE command to perform the interactive deletions, responded "Y" to RELATE's prompt after the primary key of the first record was displayed, resulting in its deletion, then responded "A" to the prompt after the second record was displayed. This resulted in all remaining records being deleted.

Applying Record Layouts

RELATE is most effective where the file on which the user wishes to operate has its definition stored in a DDL dictionary or a COBOL copybook. RELATE is able to access the dictionary or copybook in order to ascertain the format of the current file. Users can then reference group or field elements in any of the actions they wish to perform. Additionally, comparisons between files can now be made at the field level, rather than the record level, meaning that discrepancies can be more readily identified.

An example of this follows:

```
Relate> CLEAR ALL
Relate> OPEN EGFKS FORUPDATE
Relate> INFO FORMATS DDL *
   Record(s):     EGFKS-REC
                  EGFKSB-REC
   Definition(s): EMPLOYEE-REC
Relate> SET RECORD FORMAT DDL EGFKS-REC
Relate> INFO RECORD
  01 EGFKS-REC.
    02 PKEY.
      03 EMPLOYEE-ID.
        04 JOIN-YEAR                         PIC 9(2).
        04 SEQ-NUM                           PIC 9(4).
      03 COMPANY-CODE                        BINARY 16 UNSIGNED.
    02 NAME                                  PIC X(16).
    02 DEPT                                  PIC X(8).
    02 CHARACTERISTICS.
      03 FULL-TIME                           PIC X.
      03 PART-TIME-INFO.
        04 HOURS-PER-WEEK                    PIC 9(2).
      03 FULL-TIME-INFO                      REDEFINES PART-TIME-INFO.
        04 HOLIDAY-ENTITLEMENT               PIC 9(2).
    02 FILLER                                PIC X.
    02 LAST-UPDATE                           BINARY 64.
    02 NUM-SKILLS                            BINARY 16 UNSIGNED.
    02 SKILLS-MATRIX                         PIC X(8)
                                             OCCURS UPTO 8 TIMES
                                             DEPENDING ON NUM-SKILLS.
Relate> INSERT JOIN-YEAR = 92, SEQ-NUM = 1, COMPANY-CODE=99, NAME= "FRED", FULL-TIME = "Y"
Relate> INSERT JOIN-YEAR = 92, SEQ-NUM = 2, COMPANY-CODE=99, NAME= "BILL", FULL-TIME = "N"
Relate> UPDATE HOURS-PER-WEEK = 5 WHERE NAME BEGINS-WITH "BILL"
1 Records updated
Relate> SET LISTING DISPLAY EXPANDED
Relate> LIST FIELDS JOIN-YEAR, SEQ-NUM, NAME
PKEY:...   Join-year = 92,  Seq-num = 1,  Company-code = 99
   Data:   JOIN-YEAR =                 92
           SEQ-NUM =                   1
           NAME =                      "FRED "
PKEY:...   Join-year = 92,  Seq-num = 2,  Company-code = 99

           SEQ-NUM =                   2
           NAME =                      "BILL "
2 Records listed
```

In this example the user first CLEARed ALL of the current environment settings back to their default values and then opened the file EGFKS, as in the previous example. The user then used the INFO FORMATS command to search the DDL dictionary on the current subvolume for the appropriate definition name, and then applied the EGFKS-REC definition format to the currently open file using the SET RECORD FORMAT command.

Having associated a format with the current file there was no need to use the SET PRESENTATION HEX command (as in the previous example), because when a record is displayed, all fields are output in a format consistent with their type. The actual retrieved format was verified by the user entering the INFO RECORD command.

Two records were then inserted by the user, with default values being entered for unsupplied fields. The first record was then updated with some new field values.

Finally the user displayed the contents of some of the fields in all of the records within the file. Note how the output format of each field displayed was dependent on the field type.

Note also that some fields were displayed twice by the LIST command. This is because by default RELATE will output primary key values for all records displayed; this is controlled by the setting INDEX. In the case of key-sequenced files such as EGFKS, the primary key is actually formed from actual field contents, so these fields can be automatically be displayed.

The user then continued as follows with a different file, EGFKSB:

```
Relate> OPEN EGFKSB FORUPDATE
Relate> SET RECORD FORMAT DDL EGFKS-REC
Relate> DELETE WHERE JOIN-YEAR > 90
PKEY:...   Join-year = 92,  Seq-num = 1,  Company-code = 99
Delete (Y/N/A/Q)? A
2 Records deleted
```

Here the user, though working on a different file, could still use the same format as was used by EGFKS. This format had to be respecified to RELATE, however, since all record settings are cleared whenever a new file is opened.

The user requested to interactively delete a selection of records from this second file. This time when prompted with the first record's primary key, he opted to delete ALL of the records. Note, though, that this time RELATE displayed the actual component fields of the primary key. The next section actually details how the user can further customize this output so that fields appropriate to his own decision-making process are shown.

Specifying Fields to Display

In the example in the previous section, the user explicitly specified the fields whose contents were to be displayed. However, by default the LIST command (and others) displays what are known as summary fields. These are fields in the record format which have been 'tagged' with a summary attribute. This is achieved through the SET FIELD SUMMARY command, as the following example shows:

```
Relate> CLEAR ALL
Relate> OPEN EGFKS FORUPDATE
Relate> SET RECORD FORMAT DDL EGFKS-REC
Relate> SET FIELD (JOIN-YEAR, SEQ-NUM, NAME) SUMMARY
Relate> INFO RECORD PKEY
  02 PKEY.
    03 EMPLOYEE-ID.
      04 JOIN-YEAR                         PIC 9(2)
                                           SUMMARY.
      04 SEQ-NUM                           PIC 9(4)
                                           SUMMARY.
    03 COMPANY-CODE BINARY 16 UNSIGNED.
Relate> SET LISTING DISPLAY EXPANDED
Relate> LIST
═══════════════════════════════════════════════════════════════════════════════
PKEY:...   Join-year = 92,  Seq-num = 1,  Company-code = 99

            02 Pkey.
             03 Employee-id.
              04 Join-year =              92
              04 Seq-num =                1
            02 Name =                     "FRED "
════════════════════════════════════════════════════════════

   Data:   01 Egfks-rec.
            02 Pkey.
             03 Employee-id.
              04 Join-year =              92
              04 Seq-num =                2
            02 Name =                     "BILL "
═════════════════
2 Record
```

```
Relate> SET FIELD * SUMMARY
Relate> SET LISTING DISPLAY CONDENSED
Relate> LIST
================================================================================
PKEY:...   Join-year = 92,  Seq-num = 1,  Company-code = 99
   Data:   Join-year = 92,  Seq-num = 1,  Company-code = 99
           Name = "FRED            ",  Dept = " ",  Full-time = "Y"
           Hours-per-week = 0,  Last-update = 0,  Num-skills = 0
================================================================================
PKEY:...   Join-year = 92,  Seq-num = 2,  Company-code = 99
   Data:   Join-year = 92,  Seq-num = 2,  Company-code = 99
           Name = "BILL            ",  Dept = " ",  Full-time = "N"
           Hours-per-week = 5,  Last-update = 0,  Num-skills = 0
================================================================================
2 Records listed
```

In this example the user opened and set the record format as before. This time, however, he set the summary attribute on the fields whose contents were previously explicitly displayed. This was partially verified by the user in his next command: INFO RECORD PKEY. This command resulted in the display of the record format information for the record subgroup PKEY, which showed the summary attribute set on two of the fields.

The next commands entered by the operator set RELATE to use expanded listings, then LIST all the records. This time, since no field list was specified, the default action was taken which was to display all SUMMARY fields. Note that this time the EXPANDED listing option resulted in the full field hierarchy being displayed. Contrast this with the next set of commands entered.

In this case the user nominated all fields to be summary fields, using the SET FIELD * SUMMARY syntax, and specified that the listing format should be CONDENSED. Performing the next LIST command resulted in all of the fields in all of the records being displayed. In this case, because of the LISTING DISPLAY CONDENSED specification, the display of the field contents was compressed into as few screen lines as possible; this is the default method of displaying field data.

If you refer back to the record format of the EGFKS file (obtained through the earlier INFO RECORD command), you will notice that it contained a group FULL-TIME-INFO which was a redefinition of the group PART-TIME-INFO. In the last example you may also have noticed that RELATE did not display any of the fields contained in FULL-TIME-INFO. Instead RELATE assumed that the first definition encountered was the correct redefinition to apply to the data. RELATE can in fact be told how to determine which of many redefinitions to apply to the record data as it is being processed. This knowledge (which is achieved using the SET FIELD REDEFINITION command) permits the intelligent display of data in its correct format based on data content. This subject is discussed in chapter 3 in the section entitled *Applying Multiple Field* Formats.

Comparing Files

RELATE can COMPARE any two files regardless of whether or not the formats have been defined. Obviously more meaningful information can be displayed if the formats are defined - the ideal being to only display the fields that differ between the files. The following example shows this:

```
Relate> CLEAR ALL
Relate> OPEN EGFKS FORUPDATE
Relate> SET RECORD FORMAT DDL EGFKS-REC
Relate>
Relate> COPY TO EGFKSB
2 Records copied
Relate> UPDATE HOLIDAY-ENTITLEMENT = 10, DEPT = "ADMIN" WHERE NAME BEGINS-WITH "FRED"
1 Records updated
Relate> COMPARE AGAINST EGFKSB
RELATE FILE COMPARISON
   Current File:       $DATA1.REL8ISV.EGFKS
   Baseline File:      $DATA1.REL8ISV.EGFKSB
   Time of Comparison: 18-OCT-2006, 13:14:20.250
============================================================ Mismatched Record =
INDEX:     Join-year = 92,  Seq-num = 1, Company-code = 99
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - (Baseline Version)-
  Diff:    Dept = "        ",  Hours-per-week = 0
  + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + (Current Version) +
  Diff:    Dept = "ADMIN   ",  Hours-per-week = 10
═══════════════════════════════════════════════════════════════════════════════
Comparison Results:-
   Current File:........................$DATA1.REL8ISV.EGFKS
      Last Modified:....................16-OCT-2006, 10:20:34.273
   Baseline File:.......................$DATA1.REL8ISV.EGFKSB
      Last Modified:....................14-OCT-2006, 22:20:34.273
   Time of Comparison...................18-OCT-2006, 13:14:20.250
   Duration of Comparison:..............0.05 seconds

   Synchronization:.....................KEY
   Records compared from Baseline file...2
   Records compared from Current file....2
   Extra records in Current file.........0
   Records missing from Current file.....0
   Mismatched records....................1
   Total records differing ..............1
```

Here the user re-opened the original key-sequenced file EGKFS and copied all of the records within it to the baseline file EGFKSB.

The user then modified some of the fields in one of the records in the current file before comparing it against the baseline file.

Note that in this example RELATE was able to pick out exactly those fields that were modified. This is because of the types of files that were compared and the comparison strategy that RELATE chose to adopt when comparing the files. The user has several choices when configuring RELATE to compare files; these are fully discussed in chapter 3 in the section entitled *Comparing Files.*

Obey Files and Background TACL Processing

In the previous examples the user repeated the same commands several times, which gets a little tedious after a while! RELATE supports the use of command files (or OBEY files as they tend to be called). This is shown in the following examples, which carry on directly from the previous example.

```
Relate> SHOW /OUT EGCMD OBEYFORM/ RECORD
Relate> X TEDIT EGCMD
```

The first thing the user did in the above example was to output the current record settings to the file EGCMD. This was done in such a way that the file would contain the RELATE commands necessary to restore the RELATE environment to the current settings; this was achieved through the use of the OBEYFORM qualifier.

The user then executed the Guardian TEDIT program to edit the file using the X command[1]. This resulted in a background TACL process being started to execute the TEDIT command. Note that the first time that the X command is executed, there will be a delay whilst the TACL process is started. This delay will not recur in subsequent invocations of the command.

The user then edited the EGCMD file and adds the OPEN command so that it reads as follows:

```
OPEN EGFKS FORUPDATE
SET RECORD FORMAT DDL EGFKS-REC
SET FIELD * SUMMARY
```

The user then exited the editor, which resulted in being returned to a RELATE prompt. He then continued as follows:

```
Relate> CLEAR ALL
Relate> SHOW ALL

   PRIMARY RECORD SETTINGS:
      Format:     None Specified
```

1 If you are more comfortable using the older EDIT/VS editor, then you can access this instead. The syntax to do this would be "X EDIT EGCMD;XVS F;EXIT"

```
   COMPARISON SETTINGS:
      Synchronize: AUTO       Sensitivity: EXACT      Lookahead:   40
      Resync:      1          Abort:       20         Context:     ON
      Ordered:     ON         Summary:     OFF

   LISTING SETTINGS:
      Count:       ALL        Display:     CONDENSED  Unprintable: HEX
      Width:       (80)

   GENERAL SETTINGS:
      Presentation: ASCII     Dump:        OFF        Index:       KEY

Relate> X FILES EG*
EGCMD     EGDDL     EGFKS     EGFKSB    EGREF
Relate> OBEY EGCMD
Relate> SHOW RECORD

   PRIMARY RECORD SETTINGS:
      Format:      DDL        Record: EGFKS-REC  Dictionary:  $DATA1.REL8ISV
```

The user cleared all existing environment settings, then reloaded them from the file just edited. Note that only summary record settings are displayed on the screen; this is because of the potential size of the settings, and of the fact that the INFO RECORD command is available to give a more complete, consolidated view. If, as earlier, the user specified an output file in the command (for example SHOW /OUT EGCMD OBEYFORM/ RECORD), then all record settings would be placed in the output file.

The example also shows that the user entered another X command - in this case specifying a TACL macro name (note that this time there would have been no delay before the command was executed). This demonstrates that you are not limited to just running executable object files from RELATE.

Chapter Three

USER GUIDE

Introduction

As you have already discovered, RELATE allows a user to manipulate and compare records stored within any Enscribe file. The general philosophy of the operation of RELATE is that the user can gradually define more details of the file to RELATE, enabling it to process the file in an increasingly intelligent manner.

The previous chapter demonstrated some aspects of this. It showed RELATE accessing a file about which nothing was known, and how it could improve its operation once the DDL or COBOL definition was defined.

This chapter expands on this concept, and acts as a user guide to some of the more complex aspects of RELATE. It is recommended that if you are new to RELATE, you read chapter 2 first, and use it as a tutorial.

Command Shortcuts

Before you continue to learn more of RELATE's commands, you'll be pleased to learn that RELATE provides a number of features and shortcuts that reduce the amount of typing required.

First and foremost, all commands and keywords within RELATE can be abbreviated to the point of least ambiguity. For instance the 'HELP' command can be abbreviated to 'HEL' or just 'HE.' It cannot be abbreviated any further to just 'H,' because it would then be ambiguous with the command 'HISTORY.' Note that all examples in this manual use the unabbreviated syntax.

RELATE also maintains internally a list of previously entered commands. This list can be examined using the HISTORY command:

```
Relate> HISTORY
    0:    OPEN EGFKS
    1:    SET RECORD FORMAT COBOL SECTION EMP-REC OF COPYLIB
    2:    INFO RECORD
    3:    LIST
    4:    HISTORY
```

Any command in the history list can be recalled and either re-executed immediately using the '!' shortcut, or fixed, using the 'FC' command. Both methods of recall allow identification of the command to be recalled based on: its absolute position within the history list; its position relative to the end of the history list; its starting characters. The following examples all match the INFO RECORD command above:

```
Relate> FC 2
Relate> ! -3
Relate> ! IN
```

Finally, when inserting records substantially like an existing record, the user can specify the similar record as the starting point for the current insertion. This is achieved by using the LIKE qualifier to the INSERT command, as the following shows:

```
Relate> INSERT MYNAME = "FRED" LIKE WHERE MYNAME = "ANDREW"
```

Record Selection Expressions

RELATE provides a great deal of flexibility to users by allowing them to select records based on a number of criteria, using familiar SQLCI-like syntax. Examples are given in the following table:

Example Criteria	Example Command
Checking a field is exactly equal to a specified value	`LIST WHERE NAME = "FRED"`
Checking a field is not equal to a specified value	`COUNT WHERE NAME <> "FRED"`
Checking a field starts with a specified value	`LIST WHERE NAME BEGINS-WITH "FRED"`
Checking a field contains a specified value	`LIST WHERE NAME CONTAINS "FRED"`
Checking a field is less than or greater than a specified value	`DELETE WHERE NAME > "FREDERICK"`
Checking a field value matches a pattern	`UPDATE WHERE NAME ~ "FR?*K"`
Checking a record number / line number against a specified value	`DELETE WHERE $RECNUM = 54013`
Checking the record's sequence number against a specified value	`LIST WHERE $SEQNUM < 100`

RELATE automatically chooses the appropriate record keys to use, based on the information entered in the record selection expressions. For instance, consider what happens when a user enters the following command:

```
LIST WHERE f1 = "MYNAME" AND f2 > "FRED"
```

RELATE will examine the definitions for fields f1 & f2 to see if they refer to a key field. If either of them do, then RELATE will build one or more keying search paths using one or the other (or both) definitions as the basis for reading the file. Note that RELATE will only consider fields which are located at the beginning of a key: If f1 & f2 are defined within the composite group / field fkey, which is a file key, then, presuming f1 is located at the start of the field, f2 will be ignored. Writing record selection expressions that take advantage of this fact can have a major impact on how quickly the records are located. For instance the above should, in this instance, be rewritten:

```
LIST WHERE fkey > "MYNAMEFRED"
```

Another factor that affects how RELATE will access the file is whether they are using record sequence numbers, either as the default index setting (SET INDEX SEQNUM), or in the record selection expression itself (e.g. LIST WHERE $SEQNUM = 50). Doing either of these will force RELATE to read records in primary key order; resulting in RELATE reverting to a brute-force approach to finding the matching records. The impact of this can be reduced by including a range limiting term in the record selection expression (e.g. LIST WHERE f2 > "FRED" AND $SEQNUM < 50), in which case only the minimum number of records will be read that satisfy the expression.

Information on how effectively RELATE used the record selection expression to access the file can be obtained through the INFO KEYSTATS command, as the following example shows. Refer to the command reference section for further information on this command.

```
Relate> INFO KEYSTATS
Latest keying statistics for \SANFRAN.$DATA1.SSDB.FTSTKS
            Matching records found:     20
            Duplicate records:          5
            Total records read:         26
            Number of keys used:        2
            Keying success:             95%

            Elapsed time:               0.012
```

Note that RELATE does not perform any sorting of retrieved records; this is consistent with the fact that RELATE has never been intended to be a report-writer or replacement for Enform. However, it is possible in certain circumstances, to return records in order:

- If reference is only made to one key in the record selection expression, then the key's search paths will be sorted to ensure the records are read in key order. Examples are:

  ```
  LIST WHERE $RECNUM > 20000
  LIST WHERE KEY >= ""
  LIST WHERE (KEY > "Q" AND KEY < "T") OR (KEY > "C" AND KEY < "J")
  ```

- If the records are being read by sequence number (see above), then the records will always be retrieved in primary key order; of course this has a performance overhead, since all records may be read.

Applying Multiple Field Formats Using Intelligent Redefinitions

With Enscribe files it is common to redefine data elements in the record so that the physical stored data can be viewed in different logical ways. This technique is often used as a way of conserving disk space where data items are mutually exclusive to each other.

An example of this was shown in the record layout of EGFKS in the previous chapter; in this case, FULL-TIME-INFO redefined PART-TIME-INFO, and both are mutually exclusive to each other.

Another real-world example would be in a banking environment where there are multiple types of financial transactions that a customer can perform. A file that holds details of the transactions will contain multiple redefinitions to cater for all of the information needed for the different transactions. A transaction code will tell the bank's programs which type of transaction any one record refers to, and hence which redefinition to use to access the data.

Normally displaying the contents of these files would either require custom programs to be written, or would require several different Enform (or equivalent) inquiries. RELATE can intelligently process these redefinitions automatically.

It achieves this in a very simple manner. Once a record format has been specified, the user can specify an expression indicating what the conditions are for displaying the various field / group definitions within the record. When RELATE parses the record and encounters a redefined group, it will find the first group that meets its inclusion conditions, and continue its parsing with that group.

Continuing with our example from the previous chapter, the record settings would be modified as follows:

```
Relate> SET FIELD FULL-TIME-INFO REDEFINITION IF FULL-TIME = "Y"
Relate> SET FIELD PART-TIME-INFO REDEFINITION OTHERWISE
Relate> INFO RECORD CHARACTERISTICS
  02 CHARACTERISTICS.
    03 FULL-TIME                          PIC X.
    03 FULL-TIME-INFO                     IF FULL-TIME = "Y".
      04 HOLIDAY-ENTITLEMENT              PIC 9(2).
    03 PART-TIME-INFO                     REDEFINES FULL-TIME-INFO
                                          OTHERWISE.
      04 HOURS-PER-WEEK                   PIC 9(2).
Relate> LIST
══════════════════════════════════════════════════════════════════════════════
PKEY:...   Join-year = 92,  Seq-num = 1, Company-code = 99
   Data:   Join-year = 92,  Seq-num = 1, Company-code = 99
           Name = "FRED           ",  Dept = "ADMIN   ",  Full-time = "Y"
           Holiday-entitlement = 10,  Last-update = 0,  Num-skills = 0
══════════════════════════════════════════════════════════════════════════════
PKEY:...   Join-year = 92,  Seq-num = 2, Company-code = 99
   Data:   Join-year = 92,  Seq-num = 2, Company-code = 99
           Name = "BILL           ",  Dept = " ",  Full-time = "N"

══════════════════════════════════════════════════════════════════════════════
```

```
2 Records listed
```

Note how RELATE has re-organized the order of the redefinitions in the internal form of the record; this is consistent with our previous statement that RELATE will use the first definition that meets its selection criteria. Effectively the OTHERWISE clause is therefore the same as the ELSE portion of a normal IF... THEN...ELSE type expression.

The LIST command shows how RELATE has applied the formats depending on the value of the data element FULL-TIME.

It is also possible to specify that the current record format is based on the contents of previous records. This may be required, for instance, with files containing batches of related records, where each batch is preceded by an identifying ‘batch header’ record. This is achieved by marking fields as PERSISTENT using the SET FIELD command. For example:

```
SET FIELD BATCH-TYPE PERSISTENT
```

This would result in the value of the field BATCH-TYPE being ‘remembered’ whenever it was contained in a record. The value of a persistent field can then be tested in expressions by following the new keyword $PERSISTENT with the name of the persistent field. For example:

```
SET FIELD GROUP-A REDEFINITION IF $PERSISTENT (BATCH-TYPE) = "A"
```

Thus the format of the current record can now be determined by the contents of the field BATCH-TYPE contained in a previous record.

Every group or field that redefines another group or field, or is itself redefined, can use this technique; even elements contained within other elements. This has the beneficial side effect that the SET FIELD REDEFINITION expression will not necessarily have to refer to all of its parent redefinitions; because if a field is being parsed and is having its REDEFINITION expression evaluated, then all of its parents must already have been determined to be present in the record.

Another aspect of this is that when an expression is evaluated, RELATE checks stored field values without parsing the record. Therefore a field value can be checked which is not logically present in the record; this can be both good and bad. For instance, this allows a REDEFINITION expression to reference fields within the definition being tested! The next section shows an application of this.

Specifying Multiple Record Formats for a Single File

In the previous section you saw how to specify and use different formats within a single record definition in order to solve the problem of redefined data.

Users can also redefine complete records in their application by using different DDL record formats to apply to the same file. For example, in our banking scenario there might be a complete record definition for a withdrawal transaction, and another definition for a deposit transaction. The bank's applications determine the particular definition to use by looking at a portion of the physical record common to all definitions.

RELATE allows this situation by permitting the simultaneous loading of several record definitions. When RELATE loads multiple record definitions, it loads the definitions at the 02 level (below a new 01 level item it creates named REL8CSTM-REC) and automatically specifies that they are all redefinitions of each other. The user can then use the REDEFINITION clause as before to inform RELATE when each redefinition should be applied.

An example of this follows:

```
Relate> SET RECORD FORMAT DDL (REC1, REC2, RECX) (DDLVOL)
Relate> SET FIELD REC1 REDEFINITION IF REC1.RECTYPE = 1
Relate> SET FIELD REC2 REDEFINITION IF REC1.RECTYPE = 2
Relate> SET FIELD RECX REDEFINITION OTHERWISE
```

Variably Formatted Records

Sometimes records do not follow a strict fixed format specifiable by a DDL or COBOL data definition. Instead the records consist of fields (or groups of fields) that are either variable in length, or perhaps not even present in the record at all. Either situation will impact the placement of the later fields in the record.

Such data formats are common in data-communication applications, where it is desirable to reduce the physical amount of data transferred in order to optimize transmission speeds and reduce transmission costs. A common example of this is the financial services industry ISO 8583 standard, which uses a bit-map to indicate the presence of absence of data elements.

Variable format data is also found in applications with a large install-base. In such applications, data required by one installation may not be required by another. However the costs of storing and processing the unrequired data by the second installation may be huge. If the data is not present, the problem goes away. ACI's Base24 product is an example of an application that has reduced the amount of data to be processed through the use of segmented records; these segmented records use message segments of variable length and identify themselves using an ID field. Both the ID and length indicators are placed at a fixed position (relative to the start of the segment) to enable them to be processed (or ignored) by all application modules.

RELATE+ and RELATE24 both support the use of variable-format records containing fields with the following characteristics:

- Conditionally present, based on one or more other field values (PRESENT IF)
- Variable length, as controlled by a length indicator field (LENGTH IN)
- Variable occurrences, as controlled by an index field (OCCURS DEPENDING)
- Variable occurrences depending on the length of the enclosing field or message (OCCURS UNTIL)
- Truncated fields, due to the removal of trailing spaces

The following example shows RELATE24 being used to compare two Base24 files containing segmented data.

```
Relate> OPEN ACAFON

RELATE> SET RECORD FORMAT COBOBOL SEC RECDEF OF COBLIB
RELATE> SET FIELD ACCTCAF VARIABLE (PRESENT IF ACCTCAF.ID = 1, LENGTH IN ACCTCAF.SEG-LGTH.LGTH)
RELATE> SET FIELD PREAUTH VARIABLE (PRESENT IF PREAUTH.ID = 2, LENGTH IN PREAUTH.SEG-LGTH.LGTH)
RELATE> COMPARE AGAINST Q60BREL8.ACAFON

RELATE FILE COMPARISON
   Current File:       \VAN.$D2.Q60ATEST.ACAFON
   Baseline File:      \VAN.$D2.Q60BREL8.ACAFON
   Time of Comparison: 18-OCT-2006, 13:14:20.250

========================================================== Mismatched Record =
INDEX:     Num = "7411400132009902   ",  Mbr-num = 1
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - (Baseline Version)-
  Diff:    01 Caf.
            02 Seg1.
             03 Used-prd =                12
             03 Grp-prd.
              04 Ttl-wdl-prd =            53227
  + + + + + + + + + +
  Diff:    01 Caf.
            02 Seg1.
             03 Used-prd
             03 Grp-prd.
              04 Ttl-wdl-prd =            38227
==

Comparison Results:-
   Current File:........................\VAN.$D2.Q60ATEST.ACAFON
      Last Modified:....................16-OCT-2006, 10:40:34.273
   Baseline File:.......................\VAN.$D2.Q60BREL8.ACAFON
      Last Modified:....................14-OCT-2006, 22:40:34.273
   Time of Comparison...................18-OCT-2006, 13:14:20.250
   Duration of Comparison:..............0.55 seconds

   Synchronization:.....................KEY
   Records compared from Baseline file...403
   Records compared from Current file....403
   Extra records in Current file.........0
   Records missing from Current file.....0
   Mismatched records....................1
   Total records differing .............1
```

Note that variably formatted records can only be opened for read-only access.

Collapsing Definition Levels

By default, when RELATE parses a group definition for display or comparison purposes, it parses all the way down to the field level. This is not always appropriate, since this might break up an item that is normally displayed as a contiguous sequence of characters.

An example of this was shown earlier, because in our EGFKS-REC record, employee-id is defined to consist of two fields: join-year & seq-num. However the normal way to handle the employee-id is as a six character field, for example: "940123"

RELATE provides a way to handle this by allowing all of a group's component items to be collapsed to form a single composite item. This is achieved using the SET FIELD COMPOSITE command, as follows:

```
Relate> SET FIELD EMPLOYEE-ID COMPOSITE
Relate> INFO RECORD PKEY
  02 PKEY.
    03 EMPLOYEE-ID COMPOSITE.
      04 JOIN-YEAR                          PIC 9(2).
      04 SEQ-NUM                            PIC 9(4).
    03 COMPANY-CODE                         BINARY 16 UNSIGNED.

═══════════════════════════════════════════════════════════════════════════

   Data:   Employee-id = "920002",  Company-code = 99
           Name = "BILL            ",  Dept = " ",  Full-time =

═══════════════════════════════════════════════════════════════════════════
```

Comparing Files

The previous chapter introduced how the current file could be compared against a baseline file. As mentioned at that time, the user has a number of options when using RELATE to compare files.

The primary choice to make is how RELATE should synchronize itself when comparing records. The synchronization determines what RELATE tries to compare and when. The user can control this through the SET COMPARISONS SYNCHRONIZE command; the default is AUTO, which means that RELATE will choose one of three methods depending on the file type: NONE, KEY and RECORD. These methods are described in the following subsections. Note they all assume the following file contents.

Current File		Baseline File	
Rec #	Contents	Rec #	Contents
1	A	1	A
2	B	2	B
3	C	3	D
4	D	5	E
5	E	6	C
6	F	7	F

No Synchronization

The files are compared exactly in the order in which they are read. When a mismatch is detected, if a file format has been defined, RELATE will display only the fields that are mismatched. With the file contents specified above, the following sequence of actions will take place:

```
A ?= A -> Match[2]
B ?= B -> Match
C ?= D -> No match. Display differences
D ?= E -> No match. Display differences
E ?= C -> No match. Display differences
F ?= F -> Match
```

This method of synchronization is most appropriate when comparing two records directly, or when a list of records to be directly compared has already been prepared.

Key Synchronization

The files are compared by synchronizing on the primary key. Only those records with identical keys will be compared -all others are immediately considered to be mismatched.

When a mismatch is detected, if a file format has been defined, RELATE will display only the fields that are mismatched.

In the above example file, the following actions will be taken:

```
A ?= A -> Match
B ?= B -> Match
C ?= D -> No match. Display differences
D ?= ? -> Baseline record not present => No match. Display current record
E ?= E -> Match
F ?= C -> No match. Display differences (C)
? ?= F -> Current record not present => No match. Display baseline record
```

This method of synchronization is most appropriate for key sequenced files, and for other files where the primary key value is considered significant by the application that accesses the files.

Record Synchronization

The files are compared by synchronizing on records whose contents match. The primary key is totally ignored (except in order to determine the order the records are read).

Once a mismatch is detected a 'see-saw' algorithm will be employed to find the nearest matching records to the mismatch point. Mismatches won't be output until the comparison process has found a synchronization point. For formatted files, the mismatched information displayed will comprise of all current summary fields, since it is not possible to determine which one record does not match which other single record.

In the above example file, the following actions will be taken:

```
A ?= A -> Match
B ?= B -> Match
C ?= D -> No match.
D ?= D -> Match. Resync'd. Display mismatches
E ?= E -> Match
F ?= C -> No match.
F ?= F -> Match. Resync'd. Display Mismatches
```

The output for this method of synchronization is different than that for the other two methods (shown in chapter 2). For the above example it would be as follows:

```
═══════════════════════════════════════════════════════════════════
+++++++++++++
       3: C
═══════════════════════════════════════════════════════════════════
--------------------------------------------1 Mismatch(es) on BASELINE --
```

[2] The symbol '?=' is used here as shorthand for 'is compared against'.

```
        6: C
=====================================================================
Comparison Results:-
   Current File:.........................$DATA1.REL8ISV.EGF1
      Last Modified:....................16-OCT-2006, 10:40:34.273
   Baseline File:........................$DATA1.REL8ISV.EGF2
      Last Modified:....................14-OCT-2006, 22:40:34.273
   Time of Comparison...................18-OCT-2006, 13:14:22.250
   Duration of Comparison:..............0.05 seconds

   Synchronization:......................RECORD
   Records compared from Baseline file.....6
   Records compared from Current file......6
   Records matched out of order...........n/a
   Mismatched records in Baseline file.....1
   Mismatched records in Current file......1
   Consolidated mismatches between files...2
```

This shows the two records that were mismatched, and finishes by printing the statistics regarding the outcome of the comparison process. This summary shows that no checking was performed for out of order records (see the description later in this section for details on what this means), and that there was one record mismatched in each file.

It then states that there were two "consolidated mismatches between files." Consolidated mismatches are taken to be consecutive mismatched records in one or both files. For instance, if any number of consecutive records in the current file were directly mismatched against any number of consecutive records in the baseline file (including zero, as in the above example), that would only count as a single consolidated mismatch. Consolidated mismatches therefore give a slightly higher level picture of the differences between the files; they can be interpreted as meaning "mismatched blocks of records". Note that the consolidated mismatches are what the ABORT parameter uses in order to decide when to abort the comparison process.

This method of synchronization is the most appropriate when no other method is appropriate! It is used for files where the relative position of the records is significant, not the absolute position. Examples of this would be log files and edit-type files.

Other COMPARISONS settings also affect the operation when using this method of synchronization: LOOKAHEAD, RESYNC, CONTEXT & ORDERED. The impact of each of these on the above comparison actions is as follows:

LOOKAHEAD:	When a mismatch is detected the algorithm will resync at the nearest records that match each other. This parameter limits how far ahead to look from the mismatch point in order to find the resync point. Changing this in the above example would have had no effect, since the minimum value is one, which is all that was needed above.
RESYNC:	This parameter dictates how many consecutive matches must be found in order for RELATE to consider that it is back in sync following an initial mismatch condition. In the above example, changing this parameter from its default value of one to the value three, would result in no resync point being found after the first mismatch (C against D); this is because F does not match C, which would have been the third consecutive match.
CONTEXT:	This only affects the mismatch information displayed. It does not alter RELATE's comparison algorithm. When RELATE displays a mismatched block of records, if this setting is set to ON, then it will display the matching records just before and after the mismatched block. The actual number of context records displayed is the same as the RESYNC value.

ORDERED: This setting, whose default is ON, specifies that the order of records in the two files should be the same. In the case of log files, it is not unlikely that records are logged in a different order on a second run. With this parameter set to ON these would be flagged as mismatches, resulting in the user having to make a manual determination, after the comparison is complete, that the mismatched records did indeed exist in each file - they were just out of order.

If this parameter is set to OFF, then these mismatches can be automatically reconciled. This reconciliation is performed after the comparison is back in sync. Individual mismatched records in both files are individually compared against all unmatched records in the other file up to the lookahead limit. The summary statistic *records matched out of order* provides feedback to the user as to how many records are matched by this reconciliation process.

In the example above, if ordered was set to OFF, then no mismatched records would have been reported. This is because the 'C' record in the current file would have been matched against that in the baseline after the first mismatch was detected and the comparison resync'd. The value of the statistic *records matched out of order* would be one, reflecting the fact that just one record (in each file) was matched out of order.

Note that if this parameter is OFF, then CONTEXT cannot be set to ON. Additionally it is recommended that RESYNC be set to 1.

File Actions

The following table summarizes the actions that are available for different types of file:

ACTION	TYPE OF FILE					
	Edit	Unstructured	Entry-Sequenced	Relative	Key-Sequenced	SQL
Compare	Y	Y	Y	Y	Y	Y[2]
Copy	Y	Y	Y	Y	Y	N
Count	Y	Y	Y	Y	Y	N
Delete	Y	N	N	Y	Y	N
Insert	Y	[1]	[1]	Y	Y	N
List	Y	Y	Y	Y	Y	N
Update	Y	Y	Y	Y	Y	N

Notes:

1. *Operation only available at end of file*
2. *Requires RELATE/SQL module*

Batch Operation

RELATE fully supports operation within a batch environment, either through the use of command files, or through programmatic operation from TACL. This is enhanced by the fact that RELATE supports the use of Guardian DEFINES, and by the fact that RELATE will generate completion codes on termination.

In order to specify a DEFINE name instead of a physical filename, the user simply precedes the define name with an equals character; for example:

```
OPEN =EMPFILE
```

In order to determine the outcome of RELATE's operations, it returns a completion code to its process creator. This permits programmatic actions to be coded, normally within TACL, depending on the outcome of RELATE's processing.

The completion codes generated by RELATE are as follows:

COMPLETION CODE	DESCRIPTION
0	Program completed successfully. If a comparison was performed, and the EXIT qualifier was specified, this result indicates that no mismatches were detected
1	This value will only be generated if the EXIT qualifier was specified within a COMPARE command. The value indicates that the comparison process completed, but mismatches were detected.
2	This value will be detected if a user error is encountered within the program (for instance a file access failure). If a file comparison was being made, then the comparison process did not complete (for instance there were too many mismatches encountered).
3	This value is generated if an internal error occurs within RELATE. An example would be if an internal resource was exhausted (E.g. memory).
5	This value is generated if a program failure is detected by Guardian or by one of RELATE's run-time libraries. An example of this would be a process trap. If this occurs, Guardian should create a Saveabend file.

Note that unless the EXIT qualifier is used in a comparison, then during interactive use of RELATE, only the values 0, 3 or 5 can be returned to the program creator. If you explicitly leave the program by typing EXIT or CONTROL-Y, then only the value 0 will be returned.

Data Content Handlers

By default RELATE will display all fields as either strings or numbers. In some cases, this raw field data does not provide the user with meaningful information , since the data is coded in some way. Consider, for example, that an 8-byte numeric field containing a 64-bit Juliantimestamp value would store the date & time *31-MAY-2002, 12:34:56.789* as *211889633696789000!*

To address this situation, the user can specify that a field has a specific type of data content, using the CONTENT field-level setting. RELATE then provides *content handlers* that:

- Allow the raw data content of a field to be automatically translated to a more meaningful external representation when displaying the field
- Allow the user to enter new field data in the more meaningful external representation, and have it automatically translated to the field's raw data format when it is stored.

When a field containing a CONTENT setting is displayed, it is enclosed in braces '{}' regardless of the raw data format. Similarly when a user wishes to enter the data in external representation, they also enclose the data in braces. This is shown in the following example:

```
Relate> SET FIELD LAST-UPDATE CONTENT DATETIME (JULIANTIMESTAMP)
Relate> UPDATE LAST-UPDATE = {TODAY, 13:42} WHERE SEQ-NUM = 1
Relate> LIST FIELDS LAST-UPDATE WHERE LAST-UPDATE < {TODAY}
PKEY:...   Join-year = 92,  Seq-num = 1, Company-code = 99
   Data:   {18-OCT-1994, 13:42:00.000}
```

Note that the translation from the external representation to the raw field-format is performed when the command is parsed, before all file-processing commences. The following bullets highlight the issues that this presents:

- If a filter expression contains a term where the current time is obtained from a content handler, then the time is evaluated at the time the expression was entered. It does not change as the file is read.

- The sort-order of a field's external representation may not be the same as that of the field's raw data content. RELATE will always use the raw data format for establishing record keying strategies and determining whether a field value matches a term in an expression. Consider, for example, a string field mydate containing a date coded as "DDMMYY", and the user entering an expression such as LIST WHERE mydate < {23-march-1998}. If mydate is not a key field, then a record containing the date 24-march-1997 would be read by RELATE but would not be displayed. If mydate is a key field, the record may not even be read.

RELATE includes content handlers for filename conversion and date-time handling; this latter category is discussed in the next section.

Additionally, there are often cases where fields are formatted in a manner unique to a particular installation. In this case users can add code to a set of user exit routines to achieve the same result as for built-in CONTENT settings. The mechanism for adding these user exits is described in Appendix D.

The use of *custom content handlers* is seamless with the existing internal handlers.. To find information about the custom handlers in your installation, use the INFO CONTENTS command:

```
Relate> INFO CONTENTS
The following custom content handlers are installed:
   ENUMERATE
   ADD-ONE
   UNIMPLEMENTED-IN-OUT
   UNIMPLEMENTED
Relate> INFO CONTENTS, ENUMERATE
Used on numeric fields to display / enter numbers using words (e.g. 'one', 'two' etc.)
Relate> SET FIELD MYNUMBER CONTENT ENUMERATE
Relate> INSERT MYNUMBER = {TWO}
Relate> INSERT MYNUMBER = 2
```

Date-Time Handling

As described in the last section, RELATE, RELATE+ and RELATE24 can all automatically handle fields containing dates and times using the content setting DATETIME. They differ only in the number of formats supported: RELATE+ and RELATE24 can support an unlimited variety of formats, whereas RELATE only supports binary timestamps.

The binary timestamps are effectively just large integers whose interpretation is time-zone specific. RELATE can accept and display these timestamps in Gregorian format (an example of a Gregorian date is 3rd December 1994). Gregorian dates entered into RELATE or displayed by RELATE will always be displayed in Local Civil Time (LCT).

RELATE treats the timestamps in the same manner as Guardian's default action. This is summarized as follows:

- TIMESTAMP
  This is a 48 bit timestamp, always expressed in Local Civil Time. No time zone conversion is necessary between the stored value, and the value displayed by RELATE.

- JULIANTIMESTAMP
  This is a 64 bit timestamp, which uses Greenwich Mean Time (GMT) as its reference. The GMT stored values will be converted to / from the LCT equivalents when displaying / entering their values in Gregorian form.

RELATE+ and RELATE24 allow an unlimited variety of date-time formats. These can be specified as:

- Fixed-format settings (for instance YYYYMMDD, YYMMDD, JULIANTIMESTAMP)
- Freeform settings, where the user specifies layout strings indicating the placement of date-time elements in the field. The underlying storage for individual digits can be STRING, BINARY or BCD.

- User-defined format, where the user implements a user-exit to handle the particular format.

Where a field contains a 2-digit year, then RELATE+ and RELATE24 assume by default that the year is in the current century. This can be changed by specifying a Y2K-WINDOW which indicates the threshold value below which all years should be considered 21st century years. For example, with a Y2K-WINDOW setting of 80, the value 79 will be assumed to be 2079, 80 = 1980, 81 = 1981, etc. This can be particularly useful in the following situations:

- When migrating a file to become year 2000 compliant, using the COPY CORRESPONDING command RELATE+ and RELATE24 can automatically change (for instance) dates stored as YYMMDD values to JULIANTIMESTAMP values.
- RELATE+ and RELATE24 can successfully compare dates with different storage settings, enabling records using the same YYMMDD format to be compared against records using the JULIANTIMESTAMP format.

The full syntax for DATETIME CONTENT settings is given in Chapter Four under the description of the *SET* Command. The following shows some examples of date-time specifications for an 8 byte field.

```
SET FIELD LAST-UPDATE CONTENT DATETIME (YYYYMMDD)
SET FIELD LAST-UPDATE CONTENT DATETIME (STRING "%Y% %D")
SET FIELD LAST-UPDATE CONTENT DATETIME (STRING "%C-%M-%D")
```

When entering date-time values into RELATE, users can use date-time expressions that allow a great deal of flexibility. They can enter dates:

- Absolutely (E.g. last-update = {23-MAR-1998})
- Relatively (E.g. last-update = {FIRST DAY OF NEXT MONTH})

In both cases, dates and times can also be adjusted forwards or backwards (E.g. last-update = {TOMORROW + 30 days}). This adjustment can also be used in RELATE+ and RELATE24 to modify existing values of a field using the $AGE function.

```
UPDATE LAST-UPDATE = $AGE (+ 2 YEARS) WHERE LAST-UPDATE = {TODAY} AND ACTYPE = "TEST"
```

Full syntax for date-time expressions given in Chapter Four under the description of the *Standard Syntactic Definitions*.

Chapter Four

COMMAND REFERENCE

Introduction

This chapter details the complete command syntax for using RELATE, including command line options. A description of the syntax conventions used is given in Chapter 1 of this document

Start-up

Assuming that the RELATE object file resides in the current default volume / subvolume, the syntax for running RELATE is as follows:

```
Syntax:                  RUN RELATE [<file> [<open-options>] ] [; <command> [...]]

Where:
   <startup-file>        Is the initial file to be opened by RELATE. It is described below in
                         the OPEN command as <file>.
   <open-options>        Are described below in the OPEN command
   <command>             Is a valid RELATE command. If the command contains quotation
                         characters (“), then these should themselves be placed inside a
                         matching set of quotation characters, in order for them to be passed
                         through successfully from TACL.

Example:                 RUN RELATE employee
                         RUN RELATE myfile; LIST WHERE EMPNAME = “””FRED”””
```

Note that if the RELATE object code resides in the $SYSTEM.SYSTEM subvolume, then the word RUN should be omitted.

At startup RELATE will first load any installed product extensions, such as RELATE+, and display an appropriate product banner. It will then automatically search the current volume / subvolume for a command file called REL8CSTM. If found, the commands contained within it will be executed up to the first section marker (?SECTION) before any commands specified on the command line. This file will typically contain initial environment settings. Additionally, if a startup filename was specified on the command line, then after opening the file RELATE will search REL8CSTM for a matching section name, and, if found, will execute the commands within that section; Relate will use the fully qualified file name when matching against the section name, however the section name can contain wildcard specifications. For instance, a section name of *.MYFILE will match a file called MYFILE in any subvolume on any disk on any system. A section name of \\VAN.$DATA?*.MYFILE will match the file MYFILE in any subvolume on any disk whose volume name starts with $DATA on system \VAN.

RELATE will also accept various TACL PARAM messages at startup in order to override various aspects of its default configuration. These are detailed in Appendix B.

RELATE Command Overview

RELATE's commands are split into the following areas:

- **EXECUTION CONTROL COMMANDS**
  These commands are general program execution commands, common to most standard Guardian-based command-line oriented programs…
  - EXIT
  - FC
  - HELP
  - HISTORY
  - LOG
  - OBEY
  - OUT
  - VOLUME
  - X

- **ENVIRONMENT CONTROL COMMANDS**
  These commands control the various settings that affect the currently open file and the RELATE processing environment…
  - CLEAR
  - INFO
  - OPEN
  - SET
  - SHOW

- **FILE MANIPULATION ACTIONS**
  These commands perform actions on the currently open file…
  - COMPARE
  - COPY
  - COUNT
  - DELETE
  - INSERT
  - LIST
  - UPDATE

- **EXTERNAL COMMANDS**
  These commands execute other RELATE modules…
  - XSQL

Please note the following:

- Commands, like all other keywords, can be abbreviated as long as they remain unambiguous; this is described in the previous chapter in the section entitled *Command Shortcuts.*
  - Multiple commands can be placed on a single line, by separating the commands with semicolons.
  - During interactive use, commands cannot extend over a line, though they can when contained within command files, as long as an ampersand (&) is placed as the last character on all but the final line.
- Comments can be included in a RELATE command file. The comment character is the pound sign (#); it must be the first non-space character on the line, and will result in the remainder of the line being discarded. A comment character cannot appear in lines that are being continued from a previous line, since the continued lines are combined together before being processed.

The commands are described in the sections that follow. Note that these sections all use the standard syntactical definitions defined in the next subsection.

Standard Syntactic Definitions

General Definitions

The definitions listed in this section are commonly used throughout RELATE.

<field>	This is the name of a group or field defined in the current or baseline file format with no indices / occurrences specified (I.e. it is a reference to the set of groups / fields with the specified name, not a specific instance of the field within the record). To ensure no ambiguity, field hierarchy can be specified by separating level identifiers with periods.
	Examples:
	EMPLOYEE-NUM
	EGFKS-REC.EMPLOYEE-ID.EMPLOYEE-NUM
<field-occ>	This is the name of a specific occurrence of a group or field defined in the current or baseline file format. To ensure no ambiguity, field hierarchy, including field indices, can be specified by separating level identifiers with periods.
	Examples:
	EMPLOYEE-NUM
	LANGUAGES-MATRIX[1]
	MYREC.ARRAY[2].ELEMENT[3]
<field-pattern>	This is a pattern matching the name of one or more unindexed groups or fields defined in the current or baseline file format. When using field patterns, all levels of the required field's hierarchy must be specified, and be separated by periods, and the top-level must be specified as an asterisk. Valid examples are as follows:
	* = Any top-level fields (ie only the record
	.group.prefix? = Any field beginning with 'prefix' in the
	..prefix?* = Any field beginning with 'prefix' in any second level group
<identifier>	This is a sequence of alphanumeric characters and hyphens. The first character must be alphabetic.
<recnum>	This is the physical numeric index (i.e. primary key) of a record in a key-sequenced, relative or unstructured file.
<linenum>	This is the physical numeric index (i.e. line number) of a line in an edit-type file; this can include up to three decimal places.
<seqnum>	This is the logical numeric index of a record relative to the first record in the file; this can be used with any file type
<pos-int>	A positive integer literal expressed in decimal, with maximum value 32767.
<num>	A signed numeric literal, expressed in either decimal or hexadecimal. E.g.: 24, 100, %H64.
<string>	This is a string of characters enclosed by double quotes. Note that the quoted string can contain special character sequences to represent non-printable characters (except the value binary 0). The following sequences, which always start with a backslash, are accepted:
	\n - Insert newline character (%H0A)
	\r - Insert carriage return charact
	\" - Insert double-quote character
	\\ - Insert the backslash character

	Note that if you are referencing a Compaq Himalaya system name in a string literal, then you will have to precede it with two backslashes (E.g.: "\\SUPPORT.$SYSTEM.SYSTEM.TACL")
<content-value>	This is any data contained within braces '{}', up to the matching close parenthesis. It can be used when a CONTENT setting has been specified for the field, and allows data to be entered by the user in the same form that it is displayed, with RELATE (or a user-provided exit routine) handling the conversion to the internal format of the data. Note that no binary data can be inserted using character sequences prefixed by a backslash. Any backslash character will result in the character following it being treated as a literal (eg '\\\\')

Date / Time Handling Definitions

RELATE has sophisticated date and time handling capabilities. The syntax for specifying these is as follows:

```
   <date-time-spec> ::=   { <date-spec> [,] <time-spec> } [<dt-adjustment>]
                          { <date-spec>                 }
                          { <time-spec>                 }

   <date-spec> ::=        { <abs-date-spec>               }
                          { TODAY                         }
                          { TOMORROW                      }
                          { YESTERDAY                     }
                          { {FIRST} DAY OF {THIS} {MONTH} }
                            {LAST }        {LAST} {YEAR }
                            {NEXT}

   <abs-date-spec> ::=    {<delement>} [ { '/' } <delement> ] [...]
                                         { '-' }

   <delement> ::=         {<day>  }
                          {<month>}
                          {<year> }

   <time-spec> ::=        { <abs-time-spec> }
                          { NOW             }

   <dt-adjustment> ::=    { + } { <pos-int> { DAYS    } } [...]
                          { - }             { MONTHS  }
                                            { YEARS   }
                                { <abs-time-spec>     }

   <abs-time-spec> ::=    <hour>:<minute>[:<seconds>[.<microsecond>]]

N                         - <year> must be a 4 digit nu
                          - <day> can be 1 or 2 digits
                          - <month> is a minimum 3 character month name
                          - <year>, <day> & <month> can be supplied in any order, and any
                          elements may optionally be omitted, depending on the requirements of
                          the associated field.
                          - <hour> , <minute> & <seconds> are specified  using the 24-hour
                          format. Microseconds can contain anything from 1 to 6 digits.

Examples:                 31-MAY-1963, 13:15:10
                          TODAY
                          FIRST DAY O
```

Expressions

Expressions can be used in several RELATE commands to assist in the selection of specific fields or records. The syntax for these expressions is as follows:

```
   <expression> ::=      <factor> [OR <expression>]

   <factor> ::=          [NOT] {<term>            } [AND <factor>]
                               { ( <expression> ) }

   <term> ::=            <src-item> { '>='        } <constant-value>
                                    { '<='        }
                                    { '<>'        }1
                                    { '<'         }
                                    { '>'         }
                                    { '='         }
                                    { '~'         }2
                                    { BEGINS-WITH }
                                    { CONTAINS    }

   <src-item> ::=        { <field-occ>                     }
                         { $RECNUM                         }
                         { $SEQNUM                         }
                         { $LINENUM                        }
                         { $RECLEN                         }
                         { $OFFSET ( <pos-int>, <pos-int> ) }3
                         { $PERSISTENT ( <field> )


   <constant-value> ::=  { <string>           }
                         { <num>              }
                         { <content-value>    }
                         { <const-function>   }

   <const-function> ::=  { $JULIANTIMESTAMP ( <date-time-spec> )  }
                         { $TIMESTAMP       ( <date-time-spec> )  }
                         { $PARAM           ( <identifier> )      }5

N                        1. The operator '<>' tests for inequality.

                         accepts special pattern matching characters in the string value that
                         follows it - patterns are only suitable for comparing string-type
                         data. The following pattern matching characters are supported: '?'
                         matches any single character; '+' matches one or more of the
                         preceding character; '*' matches zero or more of the preceding
                         character.
                         3. The following source items are used as follows:
                         $RECNUM indicates the Enscribe record number.
                         $SEQNUM indicates the zero-based ordinal number of the record.
                         $LINENUM is used with edit files to indicate the edit line number.
                         $RECLEN indicates the length of the current record.
                         3. The parameters to $OFFSET are a start position (Zero = first
                         byte), and a length specification (specified in bytes).
                         4. If the persistent field referenced occurs multiple times within a
                         record, then the last value encountered will be the one that
                         persists.
                         5. $PARAM allows the value of a parameter to be passed through from
                         the creating process. When passing string values, they must be
                         enclosed in triple quotes, since TACL will strip single quotes. For
                         example PARAM MYNAME “””FRED”””


E
                         dept = "FINANCE "
                         dept ~ "DEV?*" AND dept <> "DEVGROUP"
                         employee.last-update > $JULIANT
                         employee.last-update < {TODAY}
                         employee.skills-matrix[1] = "FRENCH  "
```

Note that no record parsing is performed when referencing fields in expressions, therefore the specified field may not logically be part of the record. This could happen in several ways:

- The redefinition in which the field is contained is not present in a variable format record.
- A group or field index may have an invalid value.

Field Modification Definitions

These definitions are used by the UPDATE and INSERT commands.

```
   <field-upd> ::=       <rec-item> { := } <updated-value>
                                    {  = }

   <rec-item> ::=        { <field-occ>                      }
                         { $OFFSET ( <pos-int>, <pos-int> ) }1


   <updated-value> ::=   { <string>            }
                         { <num>               }
                         { <content-value>     }
                         { <constant-function> }
                         { <update-function>   }

   <update-function> ::= { $AGE       ( <dt-adjustment> ) }1
                         { $INCREMENT ( <num>           ) }3

Notes:                   1. The parameters to $OFFSET are a start position (Zero = first
                         byte), and a length specification (specified in bytes).
                         2. $AGE allows the date / time stored in a field to be modified. This
                         is only available in RELATE+ and RELATE24.
                         3. $INCREMENT adds the specified number (which may be negative) to
                         the current value of the field.

Examples:                dept = "FINANCE"
                         employee.last-update := $JULIANTIMESTAMP (TODAY, NOW)

                         last-update := $AGE (- 1 MONTH 2 DAYS)
```

Note that no record parsing is performed when specifying fields to be updated, therefore the specified field may not logically be part of the record. This could happen in several ways:

- The redefinition in which the field is contained is not present in a variable format record.
- A group or field index may have an invalid value.

RELATE Command Descriptions

CLEAR

Clears a previously defined setting specified using the SET command, to the default value for the setting. One or more settings can be cleared at a time.

```
Syntax:                   CLEAR { ALL                     }
                                { <clear-setting> [, ...] }

Where:
   <clear-setting> ::=    { [BASELINE] { RECORD                   } }
                                       { FIELD <clear-fspec> }
                          { COMPARISONS                             }
                          { LISTING                                 }
                          { DUMP                                    }
                          { INDEX                                   }
                          { PRESENTATION                            }

   <clear-fspec> ::=      <field-list> [ { COMPOSITE     } ]
                                         { CONTENT       }
                                         { CORRESPONDING }
                                         { HEADING       }
                                         { IGNORE        }
                                         { INCLUDE       }
                                         { PERSISTENT    }
                                         { PRESENTATION  }
                                         { REDEFINITION  }
                                         { SUMMARY       }
                                         { VARIABLE      }

   <field-list> ::=       { <field-id>            }
                          { ( <field-id> [, ...] ) }

   <field-id> ::=         { <field-pattern> }
                          { <field>         }

Notes:                    - Clearing the record settings automatically clears any field
                          settings in effect.
                          - If no field-level attribute is specified to be cleared, then all
                          attributes are cleared for the field.
                          - Specifying either INCLUDE or IGNORE has the same effect.
                          - Individual VARIABLE field settings cannot be removed.
Examples:
                          CLEAR ALL
                          CLEAR RECORD, COMPARISONS, BASELINE RECORD
                          CLEAR FIELD empname
                          CLEAR BASELINE FIELD empname CONTENT
                          CLEAR FIELD (empname, empid, compid) SUMMARY
```

COMPARE

Compares the specified records in the current file against those in another *baseline* file, displaying the results either on the current output device, or on a nominated temporary output device.

If record formats have been defined for the files, then a field by field comparison will take place. This allows nominated fields to be completely ignored and the case of text in nominated fields to be ignored. It also permits the comparing of records with different record formats.

A number of strategies are available for detecting mismatches between the files. The appropriate one to use, which is specified in the SET COMPARISONS command, will depend on the type of file, and the type of data in the file.

Once a mismatch is detected, then the amount of information displayed depends on the type of file, and the current environment settings. It is possible to just display those fields that differ between the files.

Refer to the section in Chapter 3 entitled *Comparing Files* for more information.

```
Syntax:                   COMPARE [/<c-qual>/] AGAINST <baseline> [WHERE <expression>]

Where:
   <c-qual> ::=           { OUT <out-file> } [, ...]
                          { EXIT           }
   <out-file>             Is a temporary new output location for the output from this command.
                          If specified then a summary of the results of the comparison will
                          still be displayed on the current output device.
   <baseline>             Is the name of the file to be compared against.
   <expression>           Is defined in the standard definitions.

Notes:                    - If the EXIT qualifier is specified, then the RELATE program will
                          terminate immediately after the comparison has been completed. In
                          this case RELATE will also return a completion code to the program
                          that started it, allowing it to programmatically take action based on
                          the outcome of the comparison process. These codes are as follows:
                              =0:  Comparison completed successfully with no mismatches
                              =1:  Comparison completed successfully, but mismatches were
                                   detected
                              >=2: Comparison terminated prematurely, or perhaps never even
                                   started
                          - If the EXIT qualifier is not specified, then the completion code
                          returned to the program creator might not reflect the true outcome of
                          the comparison.

Examples:

                          COMPARE /EXIT/ AGAINST ORIGFILE WHERE IDENT = "TEST"
```

COPY

Copies the specified selection of records to a new file at the specified location.

Normally RELATE copies records in their entirety. RELATE+ and RELATE24 additionally allow records to be copied on a field-by-field basis, automatically converting fields with different formats.

When performing this *COPY CORRESPONDING* command, the target file of the copy operation uses the BASELINE record format. Fields (or composite fields), are copied sequentially from one file to the other on an individual basis. Where the field formats differ between the two files, RELATE will first convert the field data in the primary file to an intermediate (displayable) format, using any CONTENT settings in effect. It will then convert this intermediate data into the format required by the baseline file using any CONTENT settings in effect for that file.

This mechanism allows easy re-formatting and conversion of data, for instance from a composite group containing the date in YY-MM-DD format, to a single field containing the field in a binary JULIANTIMESTAMP value.

Where there are extra fields in one file compared to the other, the field setting CORRESPONDENCE NONE can be used. This prevents the field being copied *from* (when fields are being deleted) or *to* (when fields are being inserted); in the latter case, the field will be defaulted as appropriate to the data type and CONTENT settings.

```
Syntax:                   COPY {TO               <new-file>} [<rec-loc>] [!] [<where-expr>]
                               {CORRESPONDING TO <baseline>}

Where:
   <new-file>             Is the name of another existing file.
   <baseline>             Is the name of another existing file whose format is specified by the
                          current BASELINE settings.
   <recnum>               Is defined in the standard definitions
   <where-expr> ::=       WHERE <expression>
   <expression>           Is defined in the standard definitions.

Notes:                    - If the bang (!) is specified, then existing matching records in the
                          new file will be replaced.
                          - Where appropriate for the file type, the records can be copied to a
                          specific numbered record location in the target file, to the FIRST
                          available slot, or to the END of the file. If the AT qualifier is
                          omitted, then by default the records will be appended to the end of
                          the target file.

Examples:

                          COPY TO EGFKSB WHERE SEQ-NUM > 5 OR JOIN-YEAR = 90
                          COPY TO BACKUP AT FIRST WHERE $RECNUM < 5
                          COPY CORRESPONDING TO Y2KFILES.ACNTFILE
```

COUNT

Counts the number of records in the specified selection.

```
Syntax:                   COUNT [WHERE <expression>]

Where:
   <expression>           Is defined in the standard definitions.
```

```
Examples
                          COUNT
                          COUNT WHERE JOIN-YEAR = 90 AND DEPT ~ "DEV?*"
```

DELETE

Deletes the specified selection of records from the current file, if supported for this type of file.

Deletion can be performed interactively following the display of all or part of each selected record, or can be performed totally automatically.

```
Syntax:                 DELETE [!] [WHERE <expression>]

Where:
   <expression>         Is defined in the standard definitions.

Notes:                  - If the bang (!) is not specified, then the operator will be
                        prompted before each record is deleted. The contents of the prompt
                        will be determined by the current RECORD SUMMARY settings. The
                        options when prompted are as follows:
                            Y: YES  - Delete this record.
                            N: NO   - Do not delete this record.
                            A: ALL  - Delete this record and all remaining records in the
                                      selection.
                            Q: QUIT - Do not delete this record, and skip all other
                                      remaining records in the selection.

Examples:
                        DELETE
                        DELETE ! WHERE $RECNUM < 20 OR EXPIRED = "Y"
```

EXIT

Stops the RELATE program, and returns the operator to the initiating program. This can also be achieved at any prompt by pressing CTRL-Y.

The RELATE program always returns a termination code to its creator when the program stops. If RELATE stops as a result of executing this command the termination code will always be 0.

```
Syntax:                 EXIT
```

FC

This command, which is only available during interactive operation, enables the user to FIX a previously entered command, using standard methods.

Syntax:	FC [<command-to-recall>]
Where: <command-to-recall>	Identifies a previously entered command in the history buffer, and can take several forms: - A positive integer indicates that the command at the specified location in the history buffer is to be recalled. - A negative number indicates a position in the history buffer relative to the current command is to be recalled (e.g. -1 = the previous command). - Any other text will result in the most recently entered command that matches the text being recalled.
Notes:	- If no <command-to-recall> is specified, the last command entered will be used.
	- If the command to be recalled is to be immediately re-executed without being fixed, then enter the command shortcut '!' instead of FC. - Use the HISTORY command to view the contents of the history buffer.
Examples:	FC FC SET FC -5 ! 10 ! LIST

HELP

Displays help text on a requested topic, or displays a command summary.

```
Syntax:                   HELP [ {<command>   } ]
                                 {<production>}

Where:
   <command>              Is a valid RELATE command.
   <production>           Is a production documented in the syntax description of a relate
                          command (For example, <production> is itself a production!)

Notes:                    - If no production or command is specified, then a list of available
                          commands will be displayed.

Examples:
                          HELP
                          HELP HELP
                          HELP <out-file>
```

HISTORY

Displays the contents of the command history buffer. These saved commands can be recalled either for re-execution, using the ‘!’ shortcut, or for modification, using the ‘FC’ command.

Syntax:	HISTORY [<pos-int>]
Where:	
<pos-int>	Is defined in the standard definitions. If not specified, then the value 10 is used.
Examples:	
	HISTORY
	HISTORY 20

INFO

Displays information about:

- The record format of the current file
  - Available record formats in a DDL dictionary or COBOL copybook
  - The effectiveness of the keying used in the latest file access command
- The custom content handlers installed in this copy of RELATE.

```
Syntax:                   INFO [/OUT <out-file>/] {CONTENTS [, <custom-keyword>]         }
                                                  {FORMATS {DDL <ddl-listing>     }      }
                                                           {COBOL <cobol-listing> }
                                                  {KEYSTATS                              }
                                                  {[BASELINE] RECORD [<field>] [, DETAIL]}

Where:
   <out-file>             Is a temporary new output location for the output from this command.

   <custom-keyword>       If specified, indicates that the help information for the named
                          custom content handler be displayed.

   <ddl-listing> ::=      {<rec-id>} [ ( <dict-vol> ) ]
                          {   *    }

   <cobol-listing> ::=    [<cdef-pattern>] [<csect-pattern>] {OF} <cfile>
                                                             {IN}

   <cdef-pattern> ::=     DEF {<cdef>} [FROM]
                              {  *   }

   <csect-pattern> ::=    SECTION {<csect>}
                                  {   *   }

   <field>                If specified, indicates that the format information displayed is to
                          be confined to the named group or field within the record.

Notes:                    - In situations where the format of the baseline file in a comparison
                          is different to the format of the current file, then the keyword
                          BASELINE can be used to display the format defined for that file.

Examples:
                          INFO RECORD
                          INFO BASELINE RECORD EMPLOYEE-ID, DETAIL
                          INFO KEYSTATS
```

INFO CONTENTS Subcommand

This command will display either a list of the custom content handlers installed in this copy of RELATE, or more detailed information about a specific custom content handler, as the following example shows:

```
Relate> INFO CONTENTS
The following custom content handlers are installed:
   ENUMERATE

   UNIMPLEMENTED-IN-OUT
   UNIMPLEMENTED
Relate> INFO CONTENTS, ENUMERATE
Used on numeric fields to display / enter numbers using words (e.g. 'one', 'two' etc.)
```

Custom content handlers are used in exactly the same way as RELATE’s internal content handlers (DATETIME etc.), but they allow manipulation of field data that is in a format unique to a particular installation. Refer to Appendix D for information on how to add new custom content handlers to your installation.

INFO FORMATS Subcommand

This subcommand allows a record or definition name to be located from a DDL dictionary or COBOL copybook where its name is not exactly known. Using the INFO FORMAT command the user can list out all definitions / records that start with a given set of characters.

```
Relate> INFO FORMATS DDL *
   Record(s):     ATR-ES
                  EGFKS-REC
                  EGFKSB-REC
   Definition(s): ALL-TYPES-DEF
                  EMPLOYEE-ID
                  EMPLOYEE-REC

Relate> INFO FORMATS COBOL DEF EMPLOYEE OF DDLCOB
    EMPLOYEE-ID
    EMPLOYEE-REC
```

INFO KEYSTATS Subcommand

This command will display statistics regarding the last file access operation. An example of the information displayed is as follows:

```
Relate> INFO KEYSTATS
Latest keying statistics for \SANFRAN.$DATA1.SSDB.FTSTKS
            Matching records found:     20
            Duplicate records:          5
            Total records read:         26
            Number of keys used:        2
            Keying success:             95%
            Duplication:                23%
            Elapsed time:               0.012
            CPU processing time         0.01
```

This shows that two key search paths were used in order to read the file. This resulted in 26 records actually being read, of which 5 were duplicates (returned by more than one search path). Of the remaining 21 unique records read, only 20 fully matched the record selection expression. The total time taken to process the records was 0.012 seconds, and the amount of CPU time spent processing them was 0.01 seconds.

The 'keying success' is a measure of how well the user's command could be satisfied by reading the file via its keys. It measures the ratio of matching records to unique records read (in this case 100 x 20/21). The closer this is to 100%, the better.

The 'duplication' ratio is a measure of how much redundancy there was to retrieve the records, due to records being read more than once. It is calculated as the ratio of duplicate records to unique records (in this case 100 x 5/21). The closer this is to 0% the better.

INFO RECORD Subcommand

Displays information about all or part of the current or baseline record format. The format will be displayed in field order, along with any attributes for the field defined in the SET FIELD command. Additional information will be displayed when the DETAIL option is used, such as the field offset information.

Note that the field order changes based on the RECORD REDEFINITION settings.

To see just the RELATE field attributes that have been defined for the record use the SHOW command.

INSERT

Inserts a new record with specified field values into the current file.

The specified field values will be placed either into a newly initialized blank record, or into a record template obtained by copying an existing record from the file (INSERT … LIKE).

When using a blank record, the manner in which the record is initialized depends on whether a RECORD FORMAT has yet been defined.

If no format has been defined, then the record will be set to all spaces, and the length of the record will be set to the record size of the current file.

If a format has been defined, then the fields will be initialized individually, taking into account the current RECORD REDEFINITION settings in effect.

The record length will also be determined by these same settings. The only way the record length can be changed is if a field which is the target of an OCCURS DEPENDING clause has its value explicitly changed. Take, for example, the case where the field NUM-OCCURS is the target of an OCCURS-DEPENDING clause, and is the sole field within the group OCCURS-GRP. Explicitly modifying the field NUM-OCCURS will change the size of the record. Explicitly changing the value of the group OCCURS-GRP, even though it implicitly changes the value of NUM-OCCURS, will not affect the size of the record.

```
Syntax:                   INSERT <field-upd-list> [AT {<recnum>}] [LIKE [WHERE]<expression>]
                                                      {FIRST   }
                                                      {END     }

Where:
   <field-upd-list>       Is defined in the standard definitions
   <recnum>               Is defined in the standard definitions

Notes:                    - Where appropriate for the file type, insertions can take place at a
                          numbered record, at the FIRST available slot, or at the END of the
                          file. If the AT qualifier is omitted, then by default the insertion
                          will take place at the end of the file.
                          - If the insertion is to take place at a numbered record, then the
                          record should not already exist in the current file.
                          - When inserting records LIKE another existing record, then the
                          <expression> must only match a single record.

Examples:

                          INSERT LAST-UPDATE = $JULIANTIMESTAMP (TODAY, NOW)

                          INSERT NUM = $INCREMENT (1) LIKE WHERE RECTYPE = "TEST-BASELINE"
```

LIST

This command lists the contents of the specified selection of records either to the current output device or to the output device specified in the command.

The user can either explicitly specify on the command line the portions of the record to be displayed, or can have RELATE determine from the current environment settings the information to be displayed.

If the user chooses the former, then he must state either the offsets within the record to be displayed or the group or field names. In both cases RELATE will display exactly the areas of the record requested. It will not display component fields or parse the record to determine whether the field is actually present based on the current RECORD REDEFINITION settings.

If the user does not explicitly specify the portions of the record to be displayed, then the information displayed is based on the current environment settings.

If no RECORD FORMAT is defined, then the entire record contents will be displayed in either ASCII or HEX (as determined by the PRESENTATION setting)3.

If a record format is defined, then the information displayed is determined by the RECORD SUMMARY settings. If no summary fields have been specified, then only the primary keys of the records fields will be displayed. Otherwise RELATE will parse the record based on the RECORD REDEFINITION settings, and display any summary fields encountered. This allows the intelligent display of records based on record contents.

Note that the operation of the LIST command can be further modified using the SET LISTING commands; these settings allow the user to configure how much information is to be displayed.

```
Syntax:                  LIST [/<l-qual>/] [<l-options>] [WHERE <expression>]

Where:
   <l-qual> ::=          OUT <out-file> [ CSVFORM ]
   <l-options> ::=       FIELDS {<rec-item> [, ...]} […]
                                {SKIP <num>       }
   <out-file>            Is a temporary new output location for the output from this command.
   <num>                 Is a positive integer.
   <rec-item>            Is defined in the standard definitions.
   <expression>          Is defined in the standard definitions.

Notes:                   - By default only the SUMMARY fields specified in the environment
                         settings will be displayed. This can be overridden by explicitly
                         specifying the areas of the record to display using the FIELDS
                         clause.
                         - The SKIP qualifier will prevent RELATE from listing the first <num>
                         matching records.
                         - The CSVFORM qualifier to the OUT file specification will result in
                         the output being produced in comma-separated value format, which is
                         suitable for processing by many PC applications. The output file
                         created is a binary file containing embedded carriage-return / line-
                         feed pairs at the end of every line; this binary format overcomes the
                         issue of limited line-length encountered with Guardian edit files.
                         Additional notes:
                            * All values are output as double-quoted strings, regardless
                              of their numeric type or any content settings in effect
                            * Any field formatting errors will result in fields with the
                              characters **??** appended to them
                            * The output values will honor the current LISTING UNPRINTABLE
                              setting
                            * All embedded double-quotes will be doubled up
                              (eg ab"cd"ef  -> ab""cd""ef)
                            * If the INDEX setting is not set to OFF, then the first entry
                              in each output row will contain the number of the row.
                            * This feature is only available in RELATE+ and RELATE24
```

3 If a RECORD FORMAT has been specified and the DUMP setting is set to ON, then the entire record contents will also be displayed in HEX or ASCII in addition to the areas determined from the environment settings.

```
Examples:
                        LIST
                        LIST WHERE COMPANY-CODE = 99
                        LIST /SKIP 10/ FIELDS JOIN-YEAR, SEQ-NUM WHERE COMPANY-CODE <> 99
```

LOG

Duplicates program input and output to the specified output file.

Syntax:	`LOG [ <log-file> ]`
Where: `<log-file>`	Is the name of the new output file that is to receive the duplicate output. If omitted, then the current log file will be closed.
Notes:	- If the log file already exists then the output will be appended to it.
	- Only one log file can be in use at a time.
	- The output of any secondary program executed via the 'X' command will only be sent to the current OUTPUT device – not the log file.

Examples:	`LOG MYLOG`
	`LOG`

OBEY

This command allows RELATE commands within another file to be executed. Nested command files are supported.

Commands in a command file can be grouped into sections, allowing individual sections to be OBEYed if required. Each section is identified by the keyword ?SECTION in column 1, followed by the section name. A section name consists of an arbitrary sequence of characters, including wildcard characters but excluding spaces and commas. This allows, for instance, each section in an obey-file to specify the SETtings for a particular file. Note though, that if a node name is included in the section name, the backslash prefix for the node name must be escaped by including an additional backslash, since the backslash character has special significance.

Examples of valid section names are:

- COMPARE-RESULTS
  - EMPFILE
- \\MYNODE.$DATA.MYSUBVOL.MYFILE

```
Syntax:                   OBEY [ / <obey-qual> / ] { <obey-file> [ ( <section-name> ) ] }
                                                   { *             ( <section-name> )   }

   <obey-qual> ::=        { ECHO  }
                          { QUIET }

Where:
   <obey-file>            Is the name of a command file containing the commands to be executed.

   <section-name>         Is the name of a specific section in <obey-file> that is to be
                          executed; the first section in the obey file that matches the
                          specified name will be executed.

Notes:
                          (and all nested obey files) will not be echoed to the current output
                          device. All command output will still be echoed normally.

                                                                                   ommands
                          that had been previously disabled using the QUIET option.

                                                                                       hat
                          the specified section is contained in the command file currently
                          being executed. This makes maintenance of the files easier, since
                          self-referential files can be renamed without regard to the contents
                          of the file.

Examples:
                          OBEY EGCMD
                          OBEY CMDFILE (MYSEC)
                          OBEY /QUIET/ CMDFILE (MYSEC)
                          OBEY * (MYSEC)
```

OPEN

Opens the specified Enscribe file, and makes it the current file.

Syntax:	OPEN [<file> [<open-options>]]
Where:	
<file>	Is the name of the file to open
<open-options> ::=	{ FORUPDATE }
	{ FORXUPDATE }
Notes:	- If no open options are specified then the file will be opened for read-only access.
	- Specifying the qualifier FORUPDATE will ensure that the file is open for shared read/write access.
	- Specifying the qualifier FORXUPDATE will ensure that the file is open for exclusive (i.e. protected) read/write access.
	- If no file is specified, then the name of the currently open file will be displayed.
Examples:	
	OPEN
	OPEN EGFKS
	OPEN EGFKSB FORXUPDATE

OUT

This command temporarily redirects output to the named file or device. If the file already exists then the output is appended to it.

Syntax:	`OUT [<out-file>]`
Where:	
`<out-file>`	Is the name of the new file / device where output should be directed.

Notes:	
	file being closed, and output reverting to the device specified at startup.
Examples:	
	`OUT`
	`OUT $S.#SPOOL`

SET

Sets one or more environment settings to a new value. Environment settings fall into one of the following categories:

- RECORD settings are used to define record structure and presentation information.
  - COMPARISONS settings are used to control how file comparisons are executed.
  - LISTING settings are used to control the listing of records.
- General settings define general operational characteristics.

```
Syntax:                 SET { [BASELINE] { RECORD <rec-spec> [, ...]              } }
                                         { FIELD <field-list> <field-spec> [, ...] }
                            { COMPARISONS <comp-spec> [, ...]                       }
                            { LISTING <listing-spec> [, ...]                        }
                            { { DUMP         {ON }    } [, ...]                     }
                                             {OFF}
                              { PRESENTATION {ASCII}  }
                                             {HEX  }
                              { INDEX        {OFF   } }
                                             {KEY   }
                                             {SEQNUM}

Where:
   <rec-spec> ::=       {FORMAT { DDL <ddl-load-spec>     }     }
                                { COBOL <cobol-load-spec> }
                        {FILTER [<expression>]                  }
                        {SUMMARY { *              }             }
                                 { <field>        }
                                 { (<field> [, …]) }

   <field-list> ::=     { <field-id>             }
                        { ( <field-id> [, ...] ) }

   <field-id> ::=       { <field-pattern> }
                        { <field>         }

   <field-spec> ::=     { COMPOSITE                             }
                        { CONTENT <content-type>                }
                        { CORRESPONDING NONE                    }
                        { HEADING <string>                      }
                        { IGNORE [CASE]                         }
                        { INCLUDE [NOCASE]                      }
                        { PERSISTENT                            }
                        { PRESENTATION { ASCII }                }
                                       { HEX   }
                        { REDEFINITION [{ IF <expression> }]    }
                                        { OTHERWISE       }
                        { SUMMARY                               }
                        { VARIABLE ( <varfield-spec> [, ...] ) }

   <comp-spec> ::=      {SYNCHRONIZE {AUTO  }  }

                                     {RECORD}
                                     {NONE  }
                        {SENSITIVITY {VALUE}    }
                                     {EXACT}
                        {LOOKAHEAD   <pos-int> }
                        {RESYNC      <pos-int> }
                        {ABORT       <pos-int> }
                        {CONTEXT     {ON }     }

                        {ORDERED     {ON }     }

                        {SUMMARY     {OFF    } }
```

```
                                     {ALL    }

   <listing-spec> ::=     { COUNT   {ALL      }  }
                                    {<pos-int>}
                          { DISPLAY {CONDENSED}  }
                                    {EXPANDED }
                          { UNPRINTABLE { HEX  } }
                                        { MASK }
                          { WIDTH <pos-int>      }

   <ddl-load-spec> ::=    { <rec-id>             } [ ( <dict-vol> ) ]
                          { ( <rec-id> [, ...] ) }
   <rec-id>               Is a valid DDL record or definition identifier.
   <dict-vol>             Is the location of the DDL dictionary containing the definition for
                          the current file. This can be specified as a DEFINE name with a class
                          of DEFAULTS.

   <cobol-load-spec> ::=  DEF <cdef> [FROM]] [SECTION <csect>] {OF} <cfile>
                                                               {IN}

   <cdef> ::=             <identifier>
                         This names the copy book definition to be loaded from the copy book
                          file. At least one of DEF name or SECTION name must be specified. If
                          a SECTION name is provided, then the search for this DEF name will
                          commence from the start of the specified section.

   <csect> ::=            <identifier>
                         This names a section of a copybook file marked by a ?SECTION
                          directive, containing the definition to be loaded.  At least one of
                          DEF name or SECTION name must be specified. If a SECTION name is
                         provided, and no DEF name, then the first 01 level definition within
                          the specified  section will be loaded.

   <cfile> ::=            <file>

   <content-type> ::=     { DATETIME (<dt-content>) }
                          { FILENAME24              }
                          { <custom-content>        }

   <varfield-spec>        Specifies how the field is used in a variable format record, as
                          described in detail below in the subsection entitled Variable-Format
                          Field Settings Descriptions. Variable format records are only
                          supported in RELATE+ and RELATE24.

   <pos-int>              Is defined in the standard definitions

   <expression>           Is defined in the standard definitions

   <custom-content>       A <custom-content> specification can be used when the local version
                          of RELATE has been customized to provide additional functionality. It
                          consists of a keyword optionally followed by parameters enclosed in
                          parentheses, to be passed to the user exit.

Notes:                    - In situations where the format of the baseline file in a comparison
                          is different to the format of the current file, then the BASELINE
                          qualifier can be used to specify separate formatting characteristics
                          for that file.
                          - The individual settings are described in detail in the following
                          subsections.
```

Record Settings Descriptions

FORMAT: The FORMAT setting defines the location of an external specification of the record format. This can be a Record or Definition specification stored within a DDL dictionary, or a COBOL data definition contained in some source file.

When using DDL definitions, multiple record or definition names can be specified, in which case RELATE will automatically load them as redefinitions of each other under a new 01 level item entitled REL8CSTM-REC. This

	enables multiple DDL definitions to be used by the same file, but to be defined separately. Use the REDEFINITION setting as normal to define when each definition should be used. Notes: • Setting a record format clears all other record settings. • When loading DDL definitions, RELATE will first attempt to find a record definition with the specified name. If that fails it will then try and locate a normal definition with the specified name.
FILTER:	The FILTER setting allows a record selection expression to be pre-defined, obviating the need for the operator to enter a WHERE clause in every action. Notes: • A FILTER setting can be set in addition to specifying a WHERE clause • The current and baseline records can have different FILTER settings • Fields with CONTENT settings are translated to their internal form at the time the expression is entered. This can have an impact on some types of expressions (for example those containing timestamps).

```
Examples:
                        SET RECORD FORMAT DDL EGFKS-REC (MYDICT)
                        SET RECORD FORMAT COBOL SECTION EGFKS-REC OF COPYLIB.DDLCOB
```

Field Settings Descriptions

The field settings can be applied to any combination of:

- A single field or group.
- A set of groups / fields *at the same level* that meet a wildcard specification.

The individual settings are described in the following table.

COMPOSITE: The COMPOSITE setting, if used, specifies that a group definition should be 'collapsed' into a single level, and treated as though it is a field.

For example, in the following, the group EMPLOYEE-ID will effectively be handled as a PIC X(6) item after having the COMPOSITE attribute turned on:

```
   03 EMPLOYEE-ID.
     04 JOIN-YEAR   PIC 9(2).
     04 SEQ-NUM     PIC 9(4).
```

CONTENT: This setting specifies the type / usage of data stored within data fields, permitting the field data to be displayed to the user and entered by the user in a format more meaningful that raw ASCII / numeric.

Field contents are converted using *content handlers*. The inbuilt content-handlers provided with RELATE are described later in the description of this SET command in the subsection entitled *Relate Content Handler*s.

Content handlers can also be written by customers themselves, and seamlessly integrated into the RELATE environment. For a list of locally installed content handlers, enter the command *INFO CONTENTS*. For information on how to write content handlers refer to Appendix D.

CORRESPONDIN
G: This setting is used with commands that take the *CORRESPONDING* option. It prevents a correspondence being established between fields in the primary and baseline records. During a COPY CORRESPONDING operation, this allows

fields to be inserted or deleted automatically. This setting is only available in RELATE+ and RELATE24.

HEADING

This setting allows a substitute field heading to be displayed instead of the default (the field or group name).

IGNORE:

The IGNORE attribute will be applied to the specified field or group and all component fields; this may override existing settings.

The IGNORE setting specifies areas of records that will be ignored during the comparison process. This is useful, for instance, when records contain timestamps that need not be compared. If the qualifier CASE is added, then only the case of characters will be ignored when comparing the field, not the characters themselves.

INCLUDE:

The INCLUDE setting specifies areas of records that will be compared during the comparison process. This is useful, for instance, when only a small subset of the record is to be compared, or when the record formats are dissimilar. If the qualifier NOCASE is added, then the case of the fields will be ignored during comparisons.

The INCLUDE attribute will be applied to the specified field or group and all component fields; this may override existing settings.

PERSISTENT:

The PERSISTENT setting signifies that RELATE should ‘remember’ the last value of the field(s) specified whilst a set of records is being processed by a user command. This value can then be tested in expressions through the use of the $PERSISTENT statement.

The value remembered persists across multiple records until another record containing the persistent field is encountered, at which point the persistent value will be updated. This feature is therefore only of benefit when processing variable format records, where the persistent field does not occur in every record. In particular the feature allows the format of the current record to be determined by a field value persisting from a previous record.

Notes:

- PERSISTENT fields have an impact on performance, since marking any field as persistent will force the file to be read sequentially.
- If a persistent field occurs more than once in a record, then the last occurrence will be the one that persists.

PRESENTATION

This setting specifies how data is to be presented to the user. If not specified, then the default action is to display the data in the format appropriate to the type of data (numeric / string) unless a CONTENT attribute heading has been specified, in which case the data is displayed in a format appropriate to the content type. The presence of a PRESENTATION attribute modifies this behavior by always displaying the raw field contents in one of the following formats:

- **ASCII** - Display item in ASCII
- **HEX** - Display item in hexadecimal

REDEFINITION

This setting controls how RELATE can intelligently determine which definition to use when displaying a record which contains redefinitions. RELATE will select the first definition that meets the selection criteria. The definitions will be tested in the order in which the REDEFINITION settings are defined. This order can be verified using the INFO RECORD command.

If no '*IF <expression>*' or '*OTHERWISE*' clauses are specified, then the

	specified field or group will ALWAYS be the one that RELATE will select.
	If an 'IF <expression>' clause is specified, then the specified field or group will only be selected if the expression evaluates to TRUE.
	If an 'OTHERWISE' clause is specified, then the specified field or group will be selected if none of the other fields or groups that have 'IF <expression>' clauses defined have been selected.
	Notes: • Refer to the section in Chapter 3 entitled *Applying Multiple Field* Formats for more information on how RELATE parses records.
SUMMARY:	The SUMMARY setting specifies the default fields / groups to be displayed when the LIST, DELETE and COMPARE commands are used. If no SUMMARY fields are specified, then only the primary key will be displayed by these commands.
VARIABLE:	Specifies how the field is used in a variable format record, as described in detail below in the subsection entitled *Variable-Format Field Settings Descriptions.*

Variable-Format Field Settings Descriptions

These field-level settings allow RELATE+ and RELATE24 to parse records that are not fixed in layout (as DDL or COBOL require), but have variable, content-dependent, format, including fields that are:

- Conditionally present, based on other field values
- Variable length, as controlled by a length indicator field
- Variable occurrences, as controlled by an index field or the length of an enclosing field
- Truncated fields, due to the removal of trailing spaces

In order to support these formats, the fixed-format records are specified using DDL or COBOL (as at present) and then RELATE commands can be executed to modify the format to make it variable. This is accomplished using the FIELD setting VARIABLE, which has the following additional syntax:

```
   <varfield-spec> ::=    { ALIGNMENT { WORD      }           }
                                      { BYTE      }
                                      { <pos-int> }
                          { LENGTH IN <field>                 }
                          { OCCURS { DEPENDING ON <field>   } }
                                   { UNTIL END OF {RECORD } }
                                                  {<field>}
                          { OPTIONAL                          }
                          { PRESENT IF <expression>           }
                          { TRUNCATE                          }
```

The individual settings are described in the following table.

ALIGNMENT:	Specifies how a field or group that follows a variable length field or group should be aligned. <pos-int> allows for more esoteric alignment schemes, for instance the value '8' would indicate quad-word (64-bit) alignment.
LENGTH IN:	Indicates that the length of the field or group is contained in the specified numeric field. This numeric field must be located either before the current field or must be offset a fixed amount from the current field.
OCCURS:	This attribute specifies that the field or group recurs a variable number of times.
	If the DEPENDING ON field is specified, then the number of recurrences is controlled by the specified index field; this is just as for normal DDL / COBOL recurring fields, except that the field or group does not have to be the last in the

record.

If the UNTIL END syntax is used, then the field or group recurs until the end of the specified parent entity is reached (the entire RECORD or a parent group with a LENGTH IN attribute specified). This attribute requires the field or group that is to recur to be the last field or group in the parent entity.

OPTIONAL: This attribute indicates that the field may not be present in the record.

The attribute can only be used with an elementary alphanumeric field or composite group, and even then the field or group must be the last member of a group whose length has already been determined (that is it either has a LENGTH IN attribute, or it is the entire record) or be followed by other OPTIONAL fields up to the end of the group.

The attribute cannot be used with the PRESENT IF, LENGTH IN and OCCURS attributes

PRESENT IF: Indicates that the current field or group is conditionally present in the record. That is, it is only present if the expression evaluates to TRUE. All fields referenced in the expression must either precede the current field or must be offset a fixed amount from the current field. If the expression references fields at or after the start of the current field, then if the current field is beyond the end of the record the expression is defined to be FALSE. This permits the common requirement to forward-reference fields in expressions.

TRUNCATE: This attribute indicates that the field may only be partially present in the record.

The attribute can only be used with an elementary field or composite group, and even then the field or group must be the last member of a group whose length has already been determined (that is it either has a LENGTH IN attribute, or it is the entire record) or be followed by other OPTIONAL fields up to the end of the group.

The attribute cannot be used with the PRESENT IF, LENGTH IN and OCCURS attributes

Notes:

- Variable format records are solely supported on files open for read-only access.
- By definition if a field is marked as VARIABLE, then its parent must also be variable. These parent settings are performed automatically internally.
- There are limitations on the fields / groups that can be marked as VARIABLE:
- o The fields / groups cannot themselves be redefined or redefine another field / group.
- o The fields / groups cannot be contained in any other group that recurs or is redefined, or that redefines another group.
- o Any field / group that is variable in length cannot be followed by another field / group that is itself redefined. This limitation can be worked around by placing such following fields / groups in their own parent group.
- In order to clear the variable format settings, use the CLEAR command.

```
Example 1:
   This example uses the following definition.

     01 EG1.
       02 F1.
         03 F1A.
           04 F1A1                    PIC X(10).
           04 F1A2                    PIC 9(2).
```

```
          04 F1A3                      PIC 9(2).
        03 F1B.
          04 F1B1                      PIC X(8).
          04 F1B2                      PIC X(8).
      02 F2.
        03 F2A.
          04 F2A1                      PIC X(10).
          04 F2A2                      PIC 9(2).
          04 F2A3                      PIC 9(2).
        03 F2B.
          04 F2B1                      PIC X(8).
          04 F2B2                      PIC X(8).
      02 F3.
        03 F3A.
          04 F3A1                      PIC X(10).
          04 F3A2                      PIC 9(2).
          04 F3A3                      PIC 9(2).
        03 F3B.
          04 F3B1                      PIC X(8).
          04 F3B2                      PIC X(8).

  #
  # Set fields F2 & F3 to be conditionally present based on other field values.
  #
  set field f2 variable (present if f2a2 = 2, length in f2a3)
  set field f3 variable (present if f3a2 = 3, length in f3a3)

  #
  # Set the trailing fields in the F2 & F3 groups to be optionally
  # present, as determined by the length of F2 & F3 groups.
  #
  set field f2b2 variable (optional, truncate)
  set field f2b1 variable (optional, truncate)
  set field f3b2 variable (optional, truncate)
  set field f3b1 variable (optional, truncate)
```

Example 2:

```
    01 EG2.
      02 F1.
        03 F1A.
          04 F1A1                      PIC X(10).
          04 F1A2                      PIC 9(2).
          04 F1A3                      PIC 9(2).
        03 F1B                         PIC 9(2).
        03 F1C                         OCCURS 4 T
          04 F1C1                      PIC X(3).
          04
      02 F2.
        03 F2A.
          04 F2A1                      PIC X(10).
          04 F2A2                      PIC 9(2).
          04 F2A3                      PIC 9(2).
        03 F2B                         PIC 9(2).
        03 F2
      02 F3.
        03 F3A.
          04 F3A1                      PIC X(10).
          04 F3A2                      PIC 9(2).
          04 F3A3                      PIC 9(2).
        03 F3B                         PIC 9(2).
        03 F3C                         OCCURS 1 T


  #
  #
  #
```

```
    set field f1 variable (present if f1a2 = 1, length in f1a3)
    set field f2 variable (present if f2a2 = 2, length in f2a3)

    #
    # Set occurrences of F1C to depend on an index field
    #
    set field f1c variable (occurs depending on f1b)

    #
    # Set F2C to recur until the end of group F2
    #
    set field f2c variable (occurs until end of f2)
```

Comparisons Settings Descriptions

SYNCHRONIZE: The SYNCHRONIZE setting determines the strategy RELATE uses to synchronize itself when comparing records within files. It can take the following values:

- **AUTO**: The strategy will be determined automatically by RELATE depending on the types of the files being compared. (This is the default setting).
- **KEY**: Keys will be matched in PRIMARY KEY order. Records with differing keys will be automatically deemed to not match. Records with identical primary keys will be compared down to the field level (if a format has been defined). If SYNCHRONIZE = AUTO, and the Current and Baseline files are both key-sequenced and have the same length primary keys, then this is the default synchronization method that will be automatically used.
- **RECORD**: Records will be read in primary-key order, but only the data contents of the records determine whether the records match, so the comparison process is not sensitive to absolute positions of records within the file. If SYNCHRONIZE = AUTO, then this is the synchronization method that will be automatically used if KEY is not appropriate.
- **NONE**: This method is similar to the KEY method, except that records are compared exactly in the order read (Primary Key order). No account is taken of the Primary Key value itself, instead records are compared

SENSITIVITY: This setting is only significant when comparing records with differing record formats. It determines the amount of latitude to give when comparing the individual fields. It can take the following values:

- **EXACT**: The size of the fields compared must match, and each byte of the fields themselves must match.
- **VALUE**: Only the external representation of the field item must match, taking into account any CONTENT settings for the fields. For string fields this means that trailing spaces are ignored. For numeric fields only the numeric value will be compared – not the internal representation. This setting therefore allows field representations to be changed but still to be compared on a one-to-one basis. For instance, a date-time field stored as YYMMDD can be successfully compared to one stored as a 64-bit Juliantimestamp.

LOOKAHEAD: When synchronizing based on record content, this setting determines within how many records the files must match again. If the files don't match again within this range, then the comparison will be aborted. The default for this is

40.

Notes:

- This setting is only significant if SYNCHRONIZE = RECORD.

RESYNC: When synchronizing based on record content and processing a mismatch, this setting determines how many consecutive matches must be found before the comparison can be considered to be back in sync. The default for this is 1.

Notes:

- This setting is only significant if SYNCHRONIZE = RECORD.

ABORT: This setting defines the number of differences that the comparison command can tolerate before aborting the comparison operation. The default for this is 20.

Notes:

- When performing comparisons based on RECORD content, consecutive records that are mismatched are consolidated together and only count as 1 difference for the purpose of this setting.

CONTEXT: When synchronizing on record content, and a mismatch is detected, this setting determines how many records are displayed. If OFF then only the mismatched records are displayed. If ON, then matching records before and after those mismatched are displayed as well, in order to place the mismatched records in context. The default for this is ON.

Notes:

- This setting is only significant if SYNCHRONIZE = RECORD.
- The actual number of context records displayed is the same value as the RESYNC setting.

ORDERED: When synchronizing based on record content, this setting permits individual out of sequence records to be automatically matched by RELATE. This out-of-sequence checking is performed once a mismatch has been detected, and the files are back in sync again. Before displaying the mismatched records, they will be compared up to the lookahead limit to see if they individually match any of the records in the other file.

The default for this ON, which means that no out-of-sequence checking will be performed.

Notes:

- This setting is only significant if SYNCHRONIZE = RECORD.
- This setting cannot be used if CONTEXT = ON, since placing records in their context is not appropriate if the records can be out of sequence.

SUMMARY This setting provides control over the display of summary fields during the comparison process, enabling the analysis of results.

The setting affects the display differently depending on whether or not the comparison process is performing lock-step synchronization (which is when SYNCHRONIZE has the values KEY or NONE).

Operation for SYNCHRONIZE = KEY or NONE:

- **ALL** - Summary information is always displayed if defined. If missing or inserted records are being displayed, and no summary information is available, then all non-filler fields will be displayed instead.

- **LIMITED** - Summary information is only displayed when displaying missing or inserted records; if no summary information is available, then all non-filler fields will be displayed instead.
- **OFF** - Summary information is never displayed. For missing or inserted records all non-filler fields will be displayed. This is the default method of operation.

Operation for SYNCHRONIZE = RECORD:

- **ALL** - As for LIMITED
- **LIMITED -** Summary information is only displayed when displaying context records; if no summary information is available, all non-filler fields will be displayed instead.
- **OFF -** Summary information is never displayed. For context records all non-filler fields will be displayed instead. (This is the default method of operation)

```
Examples:
                        SET COMPARISONS SYNCHRONI     CORD
                        SET COMPARISONS SENSITIVITY VALUE
                        SET COMPARISONS LOOKAHEAD 50
                        SET COMPARISONS RESYNC 1
                        SET COMPARISONS ABORT 30
                        SET COMPARISONS CONTEXT OFF
                        SET COMPARISONS ORDERED OFF
```

Listing Settings Descriptions

COUNT:	The COUNT setting limits the number of records that will be displayed in a single batch. The default for this is ALL, meaning that all records will be displayed consecutively. If a count limit is specified, then during interactive operation, following the display of the requested number of records the operator will be given the option to display another batch of records, or to quit displaying records.
DISPLAY:	This controls how information is formatted for display. By default it takes the setting CONDENSED, which means that the fields displayed are compressed onto as few display lines as possible, with minimal field labels displayed. If the EXPANDED option is specified, then each field is output on a single line. Additionally, in the case where the user has not specified a FIELD list, the full group or field hierarchy labels will be displayed.
UNPRINTABLE:	Unprintable characters can be either translated into a displayable meta-sequence of characters, or masked and converted to the '?' character. UNPRINTABLE can therefore have the following values: • **HEX**: Unprintable characters will be translated to Hex sequences. (This is the default setting). • **MASK**: All unprintable characters are displayed as '?' characters. This will result in the display lengths of all records being consistent.
WIDTH	Sets the right-margin for program listing output. This is normally determined automatically based on the output device, but this setting allows it to be overridden. The value is applied to both the current output file and the log file. Notes: • When displayed using the SHOW command, if the value has not been

overidden (ie it is the default) then it will be displayed in parentheses.

```
Examples:
                          SET LISTING COUNT 22
                          SET LISTING DISPLAY EXPANDED
```

General Settings Descriptions

DUMP: The DUMP setting, if set to ON, instructs RELATE to display every record in its raw unstructured form after displaying a formatted version of the record. The form of the display is determined by the PRESENTATION setting. This setting is used primarily to detect data misalignments. The default for this is OFF.

PRESENTATION : This setting determines how data from files with unknown formats is displayed, or how record dumps will be formatted. The setting can take the values HEX or ASCII – the default is ASCII.

INDEX: If set to KEY, RELATE will display the primary key of a record when displaying its contents; note that in the case of key-sequenced files this could result in the primary key being displayed twice. If set to OFF then no index information will be displayed. If set to SEQNUM, then the sequence number of the record within the file will be displayed (i.e. its position relative to the first record of the file); this is of primary benefit when listing entry-sequenced files, since it produces a numbering scheme more like Guardian's FUP utility. The default for this setting is KEY.

```
Examples:
                          SET DUMP ON
                          SET PRESENTATION HEX
                          SET INDEX OFF
```

Relate Content Handlers

Relate allows field data to be displayed and entered in a manner independent of the raw data type of the fields, with Relate automatically performing the conversion between the field-level representation and the external-world representation.. This is achieved by content handlers (provided either by Ascert or user-written), and specified using the CONTENT field-level setting.

The following content handlers are delivered with RELATE.

DATETIME: This setting can be used when the field contains date and/or time information. When the CONTENT setting is made, the user supplies additional information to indicate exactly how the date / time information is stored, as detailed in the following table. Note that RELATE+ and RELATE24 support dates and times stored in any format, whereas RELATE only allows 48-bit TIMESTAMP values and 64-bit JULIANTIMESTAMP values.

Setting this attribute has the advantage that dates and times can be automatically converted into something consistent and meaningful when displayed, and can be entered consistently by the user in either an absolute or relative manner, without regard to the storage format. Additionally, the DATETIME content settings facilitate the conversion of date / time data between files, and the comparison of differently coded date & time fields.

DATETIME content settings enable the following to be controlled:

- How the date and/or time is stored in a field
- How to constrain the display of time information for high precision timestamps
- For fields containing only 2-digit years, how the year is to be converted

into a 4-digit year (ie year plus century).

The syntax for specifying a DATETIME content is as follows:

```
Syntax:                  CONTENT DATETIME ( <dt-content> )

Where:
   <dt-content>  ::=     { STRING <dtlayout> } [, <dt-cqual> [, ...]]
                         { BINARY <dtlayout> }
                         { BCD    <dtlayout> }
                         { YYMMDD            }
                         { YYYYMMDD          }
                         { JULIANTIMESTAMP   }
                         { TIMESTAMP         }
                         { YEAR              }
                         { YEAR-OF-CENTURY   }
                         { MONTH             }
                         { DAY               }
                         { HOUR              }
                         { MINUTE            }
                         { SECOND            }
                         { <custom-content>  }

   <dt-cqual> ::=        { CONSTRAIN {DATE  }   }
                                     {MINUTE}
                                     {SECOND}


   <dtlayout> ::=
                         freeform manner how a date / time is stored
                         within a field. Refer to the description below
                         regarding Freeform Date-time Specifications for
                         more information.

   <custom-content>                                                 d
                         when the local version of RELATE has been
                         customized to provide additional functionality.
                         It consists of a keyword optionally followed by
                         parameters enclosed in parentheses, to be
                         passed to the user exit. Note that embedding a
                         custom content handler within a DATETIME
                         content setting enables the content handler to

                         dates / time formats.
```

Fixed Format Date-time Specifications

YYMMDD	Valid for any 6 byte field. Specifies that the field contains 2 digits each for the year of the century, the month and the day of month.
YYYYMMDD	Valid for any 8 byte field. Specifies that the field contains a 4-digit year, followed by 2-digits each for the month and day of month.
JULIANTIMESTAMP	Can be used on any 8 byte field. Specifies that the field contains a Guardian 64-bit Juliantimestamp value
TIMESTAMP	Can be used on any 6 byte field. Specifies that the field contains a Guardian 48-bit timestamp value
YEAR	Can be used on any numeric field. Specifies that the number indicates the full 4-digit Gregorian year number
YEAR-OF-CENTURY	Can be used on any numeric field. Specifies that the number indicates the 2-digit year of the century. (See the commentary on how 2-digit years are converted to

	4-digit years before display)
MONTH	Can be used on any numeric field. Specifies that the number indicates the month of the year (1 - 12)
DAY	Can be used on any numeric field. Specifies that the number indicates the day of the month (1 - 31)
HOUR	Can be used on any numeric field. Specifies that the number indicates the hour of the day (0 to 23)
MINUTE	Can be used on any numeric field. Specifies that the number indicates the minute of the hour (0 to 59)
SECOND	Can be used on any numeric field. Specifies that the number indicates the second of the minute (0 to 59)

Freeform Date-time Specifications

Freeform field layouts are accommodated using the STRING, BINARY and BCD keywords together with a layout specification, <dtlayout>, described below.

STRING	Indicates that the date-time is stored as string data. This can only be specified for alphanumeric fields, unless the layout indicates that only digits are present, in which case numeric string fields are also permitted. The layout specification can include literal characters.
BINARY	Indicates that the date-time is encoded as binary computational data. This can be specified for any type of field. The layout specification cannot contain any literal characters, but can contain an indication of the storage size of each date / time element.
BCD	Indicates that the date-time is stored as binary BCD digits. Can be specified for any type of field. The layout specification can include spaces, which are translated to the 4-bit value 15 (Hex 'F'). These are used for padding of odd-length elements

The <dtlayout> layout specification string contains date-time field descriptors and constants. Each field descriptor has the following syntax::

```
Syntax:

Notes:
                          layouts, in which case the size indicates how
                          many bytes are required for field storage (1, 2
                          or 4).  If omitted, the minimum will be used
                          (E.g. Day = 1 byte,  4 digit year = 2 bytes).

                          - A full list of types and qualifiers follows
                          (Note that case IS significant):
                             %D   =   2-digit day of month
                             %M   =   2-digit month, decimal
                             %Y   =   4-digit year
                             %C   =   2-digit year within century
                             %J   =   3-digit Julian day (day of year)
                             %h   =   2-digit hour
                             %m   =   2-digit minute
                             %s   =   2-digit second (00–59)
                             %fn  =   Fraction of second
                                      (Qualifier 'n' = '1' - '6')

Examples of entire valid layout strings::
                          "%2Y%2M%2D"
```

```
                              "%C-%M-%D, %h:%m:%s.%f2"
```

FILENAME24 Indicates that the field contains a Guardian C-series 24-byte filename. RELATE will convert data fields into the external (D-series) form for display, and back into a C-series format when entered by the user.

```
Syntax:                       CONTENT FILENAME24
```

SHOW

Shows the value of one or more settings specified using the SET command.

Two output formats are possible with this command.

The default short form is for interactive operation. When displaying the RECORD settings, this will only show the FORMAT settings; it will not display the potentially lengthy REDEFINITIONs etc. These can be displayed interactively in a more understandable form using the INFO RECORD command.

The other form is specified using the OBEYFORM qualifier; this option is only possible when the output has been redirected to a temporary output file. The output will be in the form of RELATE commands, and will consist of those commands needed to reset the environment to the current settings. This means that the current environment can easily be saved and restored.

```
Syntax:                   SHOW [/OUT <out-file> [OBEYFORM] /] { ALL                      }
                                                          { <show-setting> [, ...] }

Where:
   <out-file>             Is a temporary new output location for the output from this command.

   <show-setting>  ::=    { [BASELINE] { RECORD } }
                                       { FIELD  }
                          { COMPARISONS          }
                          { LISTING              }
                          { DUMP                 }
                          { INDEX                }
                          { PRESENTATION         }


Notes:                    - If the OBEYFORM qualifier is used, then the format of the output
                          will consist of RELATE commands. See the description above.

Examples:
                          SHOW ALL
                          SHOW /OUT RECCMDS OBEYFORM/ BASELINE RECORD
```

UPDATE

Updates the specified selection of existing records with the new field values. Refer to the INSERT command for information on how to insert new records into the file.

The only way the record length can be changed is if a field which is the target of an OCCURS DEPENDING clause has its value explicitly changed. Take, for example, the case where the field NUM-OCCURS is the target of an OCCURS-DEPENDING clause, and is the sole field within the group OCCURS-GRP. Explicitly modifying the field NUM-OCCURS will change the size of the record. Explicitly changing the value of the group OCCURS-GRP, even though it implicitly changes the value of NUM-OCCURS, will not affect the size of the record.

```
Syntax:                   UPDATE <field-upd-list> [WHERE <expression>]

Where:
   <field-upd-list>       Is defined in the standard definitions
   <expression>           Is defined in the standard definitions.

Notes:                    - In order to modify the value of a field relative to its existing
                          value, then a function must be used, such as $INCREMENT. Refer to the
                          section entitled Field Modification Definitions at the start of this
                          chapter for more information.

Examples:
                          UPDATE HOLIDAY-ENTITLEMENT = 10, DEPT = "ADMIN"
                          UPDATE HOURS-PER-WEEK = 5 WHERE NAME BEGINS-WITH "BILL"
                          UPDATE HOLIDAT-ENTITLEMENT = $INCREMENT (2)
```

VOLUME

This command changes the current default volume or subvolume used for filename expansion.

This new volume specification will also be passed through to the background TACL (if any is active).

Syntax:	VOLUME [<vol-subvol>]
Where:	
<vol-subvol>	Is the name of the new default volume or subvolume. If not fully qualified, then the existing default will be used to expand it. If omitted, then the default volume or subvolume will be reset to that used at startup.
Examples:	
	VOLUME VOLUME $DATA1.APPDB

X

This command allows TACL macros and programs to be executed from within RELATE. This is useful, for instance, to create or alter file characteristics (using FUP) or to directly access the operator's preferred editor when developing RELATE command files.

This is achieved by starting a background TACL process the first time that the command is executed; an initial startup delay may therefore be experienced this first time.

The user can also opt to invoke an interactive TACL session as detailed below by simply typing 'X' with no parameters; this will result in a TACL prompt being displayed. To return to RELATE the user should type EXIT, or press CONTROL-Y.

Note when using this command that RELATE and the background TACL are two distinct programs, with their own operating environments. RELATE will try and keep the two environments in sync (for instance the current default volume / subvolume), but the user can circumvent these efforts if they try hard enough.

```
Syntax:                   X [<os-command>]

Where:
   <os-command>           Is a complete TACL command line that will be passed to the background
                          TACL for execution. If omitted, then an interactive session will be
                          initiated with the background TACL, and the operator will be given a
                          TACL prompt.

Notes:                    - All text to the end of the line is treated as part of the TACL
                          command line, and will be passed through to the background process.
                          - This command ignores any temporary output locations specified with
                          the OUT command; all output will instead be directed to the output
                          device specified at RELATE's startup.

Examples:
                          X
                          X TEDIT REL8CSTM; DISPLAY L-20
                          X FILEINFO REL8*.*
```

XSQL

This command executes the RELATE/SQL program, in order to perform comparisons on SQL tables. Refer to the RELATE/SQL User Manual for further information.

Syntax:	`XSQL [/ <cl-options> /]  [ <xsql-commands> ]`
Where:	
<cl-options>	Are regular TACL command-line options that can be specified (such as an IN file, CPU number etc.).
<xsql-commands>	Are commands that will be passed directly into RELATE/SQL for execution.
Notes:	- All text to the end of the line is treated as part of the XSQL command line.
Examples:	
	`XSQL`
	`XSQL LOAD ABC START`
	`XSQL /IN CMDFILE/`

Appendix A

INSTALLATION PROCEDURES

Introduction

This appendix details how to install RELATE on your system, and lists the system management issues surrounding the installation.

System Management Issues

Before installing RELATE, your system manager will want to know various information about RELATE. This section attempts to answer some of the common questions about the product.

Disk Usage:	A full installation of RELATE requires approximately 0.5Mb of disk space. All installation files are downloaded into a single subvolume from the distribution media.
PCB Usage:	Each simultaneous user of RELATE will run their own copy of the program. They will each use a single PCB.
Security:	RELATE can run using any user-id and does NOT require the setting of the file attributes LICENSE or PROGID. The product uses standard Guardian security. We recommend that for the security of the product, ownership be given to SUPER.SUPER, and the object file's access protection altered to prevent unauthorized access to it. Note the product employs a licensing scheme to ensure access to the product is limited to bona fide customers.
Hardware Support:	The product supports the entire range of Compaq Himalaya platforms. Note that the product licensing scheme restricts access only to hardware platforms for which the license is valid.
OS Support:	RELATE will run on any D-series or higher release of the Guardian Operating System. RELATE can be run as a high-PIN process. RELATE does not use any OS specific or privileged code.
Network Support:	RELATE can access files across the Compaq Himalaya's EXPAND network.
Terminal Support:	The product runs as a conversational program, and as such does not require anything more than a TTY interface. RELATE can also be run within a SEEVIEW environment.
Software Dependencies:	None. RELATE is a completely standalone program.

Product Installation

RELATE's program and demonstration files may be supplied on several different types of media, which require different installation methods. This section details the steps required to install RELATE from magnetic tape, floppy disk or the Internet.

In all cases the files provided must be copied to an Installation Subvolume (ISV) which is where all the files associated with a particular release of RELATE are kept together. We suggest using the name RELATE.

Your supplier should have provided a product license that enables the program to run on your system. Make sure you have the license information to hand before continuing, since it may be required during installation.

Installing from Magnetic Tape

1. Restore the Product Files from tape into an Installation Subvolume. The command to do this will be something like the following:

```
RESTORE $tape, $*.*.*, vol $myvol.relate, listall, keep, myid, tapedate
```

2. Run the Installation Program INSTALL, which will prompt for any information that it requires. To run the program, make the RELATE ISV your current subvolume, and type the following:

```
RUN INSTALL
```

   If the program completes with no errors, then RELATE is successfully installed!

Installing from Floppy Disk or the Internet

1. When installing from floppy disk or the Internet, the Guardian files are contained in a self-extracting ZIP file archive. This archive file, REL8INST[4], should be transferred to an empty Relate Installation Subvolume (ISV) on your Compaq Himalaya system (e.g. using IXF or FTP). **Note when doing this that you must select a BINARY file transfer mode in your file transfer program.**
2. Change the file code of the transferred file on your Himalaya system to 100, by typing the following:

```
FUP ALTER REL8INST, CODE 100
```

3. Extract and install the files from the archive by running the program:

```
RUN REL8INST
```

   If the program completes with no errors, then RELATE is successfully installed!

Description of Delivered Files

The following list details the files contained in a normal release of RELATE. This is provided for your information, since not all of the files are required for RELATE to operate.

FILENAME	DESCRIPTION
AARLSNOT	Product Release Notes, containing new information about this release of RELATE.
EG*	A number of example files referenced by this manual.
INSTALL	A TACL command file used to control the installation process.

[4] Note that a shortcut exists when using FTP to transfer binary files to Guardian systems. In this case the Guardian file code, prefixed with a comma, can be appended to the filename prior to transfer. This file code will then be automatically applied by the Himalaya FTP server software during file transfer. For example the "REL8INST" file could be renamed to "REL8INST,100" prior to transfer; if the filename on your distribution media has this name, then the file has already been named to take advantage of this shortcut.

PROLIU	Product Licensing Utility; used to license this installation.
RELATE	The object code file for RELATE.
REL8HLPX	Help text file. May be customized for local operation.
REL8XITC	Source file of user exit stubs for customizing RELATE.
RL8LICE	License file RELATE uses to validate operation of this copy of RELATE. Note this must always reside on the same subvolume as the RELATE object code file.

Appendix B

RELATE STARTUP PARAMETERS

Introduction

This appendix details the run-time parameters that RELATE will accept at startup, and which modify aspects of its internal operation.

The syntax for specifying a PARAM from TACL is as follows:

```
Syntax:                   PARAM <param-name> "<value>"

Where:
   <param-name>           Is the name of one of the parameters described in the table below.
   <Value>                Is a value appropriate to the parameter name.
```

Startup Params

Some of the following parameters control the allocation of internal RELATE resources. Please notify your supplier if you find you have to make changes to any of the default values.

PARAM NAME	DEFAULT VALUE	DESCRIPTION
Ddl-layouts	-	This parameter allows RELATE to handle DDL dictionary versions other than those already coded specifically within the program. Contact your vendor if you feel you need to use this parameter.
Max-map-entries	1000	Maximum number of DDL / COBOL field descriptions that can be handled by RELATE. Note that Level 88's etc. which are not supported by RELATE do not affect this figure.
Parse-tree-size	500	This parameter limits how complex a user command that is entered on a single logical command line can be.

Appendix C

COMMON USER MESSAGES

Introduction

This appendix details some of the common messages generated by RELATE, together with their causes and cures.

Messages

Ambiguous field reference
Cause: A field was specified that could not be uniquely identified due to other fields having the same name.
Cure: Qualify the field name using some of its ancestor group names.

Command too complex:...
Cause: There is a limit to the complexity of a command entered on a single line.
Cure: Make the command less complex, or, alternatively, increase the size of the startup parameter PARSE-TREE-SIZE.

Conflict between CONTEXT and ORDERED settings
Cause: If ORDERED is set to OFF, then CONTEXT must be also be set to OFF.
Cure: Alter the above settings as appropriate.

Data type / value mismatch with field <field>
Cause: This could occur because a numeric value was entered for a string field, or vice-versa. Alternatively it could occur because a numeric value entered by the user could not be scaled to meet its internal representation stored in the record.
Cure: Re-enter the data, or check the record definition.

Definition too complex for internal layout map
Cause: The number of fields in the definition(s) loaded exceeded one of RELATE's table sizes.
Cure: Increase the value of the startup parameter MAX-MAP-ENTRIES.

Error updating field <field> in record
Cause: This message is generated if a numeric conversion fails when updating a numeric string field in a record.
Cure: Check the value you are entering against the field description.

Field <field> has bad value
Cause: This message is generated when reading a numeric string field, and the conversion fails due to invalid data being in the field. The conversion will be automatically aborted, and the string value displayed instead.
Cure: Check that you are using the correct record format. Consider using a PRESENTATION setting on the field to alter the default display type.

Internal consistency failure in...

Cause: An internal consistency check performed by RELATE failed.

Cure: This should never happen. If it does contact your support representative and give them full information as to what you were doing when the message was generated.

Invalid field occurrence

Cause: You have entered a field occurrence index that is less than zero or greater than the maximum value for the field.

Cure: Re-enter the correct value, or change the definition for the record.

Invalid length in numeric expression $OFFSET (<offset>, <len>)

Cause: Either an offset length of 0 was specified or one greater than 8 (the maximum integer byte size on a Himalaya system)

Cure: Change the offset length specification

License Exception #nn

Cause: This series of messages is displayed when a licensing problem is detected. The program will immediately stop running. The following list shows some of the exception codes that may be produced:

- #1: An error has been encountered with the License File. The subcode indicates the actual Guardian file system error number (for example subcode 48 would indicate a file security problem).
- #3: The license file is of an incorrect type: it should be an edit file. This could be an old license file, or one that has been transferred incorrectly to your system.
- #4: The contents of the license file are corrupt. Contact Customer Support staff, quoting the subcode displayed.
- #5: The license file is for a different product; each license file is good for only one product per enterprise.
- #6: The License has expired. Contact Customer Support for a renewal.
- #7: The License maintenance period has expired, and the product version you are attempting to run is newer than your expired maintenance contract permits. Contact Customer Support for a renewal.
- #10: Options required for the product to run successfully have not been licensed. Contact Customer Support staff, quoting the subcode displayed.
- #11: The object code filename of the product has been changed. For the product to run successfully it must have the original filename.
- #15: An internal error has occurred in the License checking software. Contact Customer Support, and inform them of the subcode displayed.

Cure: Take the appropriate action as determined by the exception code.

Mismatch between file / record size settings

Cause: This warning message is emitted when the actual record length of a file does not match the value contained in the definition that the user is using to access the file.

Cure: Verify that the definition is the correct one to associate with the file.

No format information available

Cause: An attempt was made to display settings or layout information when no format has yet been specified by the user.

Cure: Enter a RECORD FORMAT specification.

No primary key information in DDL

Cause: This informational message is generated when the user has opened a key sequenced file, but the definition specified as containing the record layout does not have primary key specifications within it. This could be because it is a normal structure definition, rather than a record definition, or because the file type associated with the record definition is not a key-sequenced file. RELATE will automatically attempt to locate the primary key fields using the primary key information contained in the file directory entry. However it will choose the first fields that refer to the primary key, which may not be the correct redefinition to use.
Cure: Specify a DDL Record Definition name when accessing key-sequenced files.

Operator <op> not appropriate for numeric item
Cause: The specified relative operator is only suitable for string fields.
Cure: Change the expression, or check the record definition.

Unable to allocate memory for...
Cause: RELATE ran out of its internal memory.
Cure: Use BIND to increase RELATE's HEAP size - contact your support representative for details on how to do this.

Variable length record format specified for an UNSTRUCTURED file.
Cause: The applied record format contains an OCCURS DEPENDING clause. When processing unstructured files, unless RELATE can treat them as containing fixed-length records, then the results will probably be unwanted. The reason is that RELATE does not pad short records when writing data, therefore a subsequent re-read of the data will result in the initial bytes of the subsequent record being retrieved in addition to the current record.
Cure: To avoid this, do not use record formats that contain the OCCURS DEPENDING clause.

Version <version> of DDL dictionary is not recognized
Cause: The version code of the dictionary is not one of those internally coded within RELATE.
Cure: RELATE can cope with this situation by using one of its startup parameters, but you should contact your support representative for information on how to do this.

Appendix D

CUSTOMIZING RELATE

Introduction

This appendix provides an overview of how to customize your copy of RELATE using user exits. These allow custom content handlers to be specified that can be used to present field data from files in a manner more understandable to the users, as well as allowing the users to enter the data in the same understandable way.

Custom Presentations

Custom content handlers, once added as user exits, are seamlessly available to the user, just as the standard internal content handlers are. In summary, the user exits that enable this are as follows:

- rel8cstm_content_in
  This user exit is called to convert data from its external (viewable / meaningful) representation to an internal representation.
- rel8cstm_content_out
  This user exit is called to convert data from an internal representation to its external (viewable / meaningful) representation.
- rel8cstm_content_set
  This user exit is called when an operator specifies they wish to use a custom content handler for a specific field. The function verifies that the setting is appropriate for the field.

More detailed information, including procedure stubs and Bind instructions are contained in the file REL8XITC on the RELATE installation subvolume.

Custom Help Text

The delivered help text is contained in the file REL8HLPX. This file can be modified if desired by the customer; full details are enclosed in the file REL8XITC on the RELATE installation subvolume. If several help text files are required, for instance to provide help in multiple languages, then the user can specify which help file to access by adding the Guardian Define name =CliHelpFilename to their TACL environment.

Appendix E

LIMITATIONS

Introduction

This appendix details the known limitations of using RELATE on Compaq Himalaya platforms.

General Limitations

- The only OUT and OBEY file types that RELATE will accept are edit-type files (code 101) and C binary files (code 180).
- Versions prior to v2.4.1 are unable to handle Guardian Format 2 files.

COBOL Definition Limitations

- Relate only directly supports the use of "non-edited" data items, which are those which can be expressed by the PIC characters '9', 'A', 'X' & 'V'. This is because these items are the ones which have correspondence back to the native DDL definitions. If data on file uses formats that require the use of edited data items, then the workaround would be to create a custom content handler in Relate to do the translation.
- Level 66 items cause an error. Level 77 items will terminate a level 01 definition, but are otherwise ignored. Level 88 items are ignored.
- USAGE INDEX is not supported, since its use is ambiguous between COBOL74 & COBOL85
- KEY specifications are not supported within OCCURS clauses.
- The following keywords are accepted but ignored:

BLANK	EXTERNAL	IS	JUSTIFIED	RIGHT	USAGE
ZERO					
CHARACTER	GLOBAL	JUST	LEFT	SIGN	WHEN

- In COBOL an identifier consists of a sequence of alphanumeric characters & hyphens:
    - 1. First and last characters can't be hyphens
    - 2. Must contain at least one Alpha or hyphen character

  RELATE's rules are slightly different, since it requires the first character to be alphanumeric, thus satisfying rule 2, but causing a possible (though rare) incompatibility.

DDL Definition Limitations

- DDL RENAMES clause is not supported
    - DDL mathematical data types are not supported (complex, float etc.)

INDEX