



# TIP Query Language

---

IP-627

Draft 2.5 - Confidential

September 2004



This edition applies to TIP Studio 2.5 and revision levels of TIP Studio 2.5 until otherwise indicated in a new edition. Publications can be requested from the address given below.

Inglenet Business Solutions Inc reserves the right to modify or revise this document without notice. Except where a Software Usage Agreement has been executed, no contractual obligation between Inglenet Business Solutions Inc and the recipient is either expressed or implied.

It is agreed and understood that the information contained herein is **Proprietary** and **Confidential** and that the recipient shall take all necessary precautions to ensure the confidentiality thereof.

*If you have a license agreement for TIP Studio or TIP/ix with Inglenet Business Solutions Inc, you may make copies of this documentation for internal use. Otherwise, you may not copy or transmit this document, in whole or in part, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of Inglenet Business Solutions Inc.*

Inglenet Business Solutions Inc

Toll Free: 1-800-387-9391  
Website: <http://www.Inglenet.com>  
Help Desk: [HelpDesk@Inglenet.com](mailto:HelpDesk@Inglenet.com)

TIP Studio, TIP/ix, and TIP/30, and are registered trade marks of Inglenet Business Solutions Inc:

This documentation occasionally makes reference to the products of other corporations. These product names may be trade marks, registered or otherwise, or service marks of these corporations. Where this is the case, they are hereby acknowledged as such by Inglenet Business Solutions Inc.

© Inglenet Business Solutions Inc, 1994-2004



# Contents

<b>TQL - TIP Query Language .....</b>	<b>1</b>
<b>Introduction .....</b>	<b>1</b>
<b>TQL Components .....</b>	<b>1</b>
Query Language.....	1
Development Environment.....	1
Runtime Interpreter .....	2
<b>Porting Your Programs to TQL.....</b>	<b>2</b>
General TQL Changes .....	2
Language Definition Changes .....	3
cvttql - TIP/30 to TQL Converter .....	4
Required Conversions.....	5
Running the Converter .....	6
Importing TQL Applications into TQL .....	7
Saved Command File Conversion.....	8
<b>TQLCC - TQL Compiler .....</b>	<b>9</b>
<b>Reserved Words.....</b>	<b>9</b>
<b>COPY Statement .....</b>	<b>11</b>
<b>Fields.....</b>	<b>12</b>
Ambiguous Field References .....	13
<b>TQL Expressions .....</b>	<b>15</b>
<b>File Definition .....</b>	<b>17</b>
<b>Record Definition .....</b>	<b>19</b>
<b>Group Items and TQL .....</b>	<b>21</b>
<b>ALLOW: Changing Fields .....</b>	<b>21</b>
<b>ALLOW: Exporting Fields .....</b>	<b>23</b>
<b>ALLOW NULL: Fields.....</b>	<b>24</b>
<b>ALLOW: GO (Auto Update).....</b>	<b>25</b>
<b>Hidden Fields.....</b>	<b>25</b>
<b>MUST ADD: Fields .....</b>	<b>26</b>
<b>Preventing Use of Run-time Commands.....</b>	<b>27</b>
<b>Record Selection - ID IS .....</b>	<b>28</b>
<b>Key Prefix.....</b>	<b>28</b>
<b>VERIFY: Fields .....</b>	<b>30</b>
<b>System Fields .....</b>	<b>31</b>

<b>TQL Program Structure .....</b>	<b>33</b>
IDENTIFICATION DIVISION .....	34
DATA DIVISION .....	36
WORKING STORAGE SECTION .....	37
TQL Statements .....	38
DECLARATIVES SECTION .....	41
DISPLAY DIVISION .....	48
REPORT DIVISION.....	55
Report Heading and Footings .....	63
Control Breaks.....	64
TQL Execution Cycle.....	65
<b>TQL Interface to DMS.....</b>	<b>66</b>
SCHEMA Definition .....	67
TQL/DMS Programming .....	67
TQL/DMS: Currency Considerations.....	73
Runtime Database Errors.....	76
<b>TQLRUN - TQL Runtime Interpreter .....</b>	<b>78</b>
<b>TQLRUN Features .....</b>	<b>78</b>
<b>Function Keys.....</b>	<b>78</b>
<b>TQL Program Execution .....</b>	<b>79</b>
<b>Multiple TQL Commands.....</b>	<b>81</b>
<b>AD HOC Modifiers .....</b>	<b>82</b>
BY.....	82
FROM.....	83
GO.....	83
IF .....	84
INTO.....	84
key-value.....	85
MOVE.....	86
ON.....	87
SORT.....	87
SUM.....	89
TO.....	89
WHERE .....	90
ADHOC Commands.....	91
Using a Predefined Display.....	91
ADD Record .....	93
CHANGE Data .....	93
CLOSE TQL Program .....	93
COUNT Records .....	94

DELETE Record .....	94
DROP Selection .....	95
END TQL Program .....	96
ENTER Several Records.....	96
EXECUTE A Saved Command .....	97
EXPORT Data .....	97
Free Format LIST .....	99
Display MORE Data .....	100
MOVE Field or Value.....	101
Display NEXT Screen of Data .....	101
OPEN New Program .....	101
Display PREV Screen of Data.....	101
PRINT Predefined Report .....	102
Free Format PRINT .....	102
RECALL a Command.....	104
RELEASE a Selection.....	104
SAVE a Command .....	105
SELECT Subset Of A File .....	106
SHOW Field Names and Selects .....	108
SORT Records .....	109
SUM Fields.....	109
UPDATE Record .....	111
Direct Execution of TQL Programs.....	112
<b>TQLMON - TQL Development Environment.....</b>	<b>115</b>
<b>TQLMON Features .....</b>	<b>115</b>
<b>TQL Development Cycle .....</b>	<b>115</b>
<b>Editing .....</b>	<b>116</b>
<b>Compiling.....</b>	<b>116</b>
<b>Templates and Program/Record Cloning.....</b>	<b>117</b>
<b>Concurrency Control .....</b>	<b>118</b>
<b>TQLMON Command line Options.....</b>	<b>118</b>
<b>Command Summary .....</b>	<b>119</b>
<b>TQL Commands .....</b>	<b>121</b>
Command Summary .....	121
AF - Add File .....	123
AS - Add Schema .....	123
C, CF, CP, CPT, CT - Compile From External Source ...	124
CD - Change Current Working Directory.....	125
COMP - Compile Existing Records and/or Programs .....	125

DEL - Delete File or Record .....	126
DP - Delete Program .....	126
DPT, DT - Delete Program or Record Template .....	127
DS - Delete Schema .....	127
E - End TQLMON Program .....	128
EDIT - Edit a Source File .....	128
HELP - Display Help Information .....	128
L - List File/Record .....	129
LP - List Program .....	129
LPT, LT - List Program or Record Template .....	129
M - Make Screen Formats .....	130
N - Define New Record .....	131
NF - Define New File .....	132
NP - Define New Program .....	133
NPT - Define Program Template .....	134
NT - Define Record Template .....	135
O - Open Program .....	135
P - Print File/Record .....	136
PP - Print Program .....	136
PPT, PT - Print Program or Record Template .....	137
RUN, R - Run TQL Program .....	137
S - Summarize File/Record .....	138
SMFILE - Invoke SMFILE .....	138
SP - Summarize Programs .....	139
SPT, ST - Summarize Program or Record Templates ....	139
SS - Summarize Schemas .....	140
TFD - Invoke TFD .....	140
U - Update Record Definition .....	141
UC - Update Control Record .....	141
UF - Update File Definition .....	146
UP - Update Program Definition .....	146
UPT, UT - Update Program or Record Template .....	147
W - Write File/Record Definition to Source File .....	147
WP - Write Program Source to File .....	148
WPT, WT - Write Program/Record Template to Source File .....	149
XF/XFC - Cross Reference Files .....	150
XP - Cross Reference Programs .....	150
XR/XRC - Cross Reference Records .....	151
<b>TQLADMIN - TQL System Administration .....</b>	<b>152</b>
<b>TQLADMIN Features .....</b>	<b>152</b>



<b>Initializing the TQL Control File.....</b>	<b>152</b>
<b>Checking the TQL Control File .....</b>	<b>154</b>
<b>Rebuilding the TQL Control File.....</b>	<b>155</b>
<b>Cleaning the TQL Control File .....</b>	<b>156</b>
<b>Clearing Edit Locks .....</b>	<b>157</b>
<b>Setting Options in TQLADMIN .....</b>	<b>158</b>
<b>Converting Saved Command Files .....</b>	<b>159</b>
<b>Exiting TQLADMIN .....</b>	<b>160</b>
<b>TQLSVE - TQL Saved File Maintenance.....</b>	<b>161</b>
<b>Installing TQLSVE.....</b>	<b>161</b>
<b>Running TQLSVE .....</b>	<b>161</b>
<b>TQL Example Programs.....</b>	<b>163</b>
<b>Inventory/Order Example .....</b>	<b>163</b>
<b>ORD Program Description .....</b>	<b>166</b>
<b>INVOICE Program .....</b>	<b>167</b>
<b>ANSI COCOL-85 Specifications.....</b>	<b>170</b>
<b>Qualification .....</b>	<b>170</b>
<b>Reference Modification .....</b>	<b>170</b>
Function.....	170
General Format .....	170
Syntax Rules .....	171
General Rules .....	171
Identifier.....	172
<b>Index .....</b>	<b>173</b>



# TQL - TIP Query Language

---

## Introduction

TQL is an interactive facility that allows you to create flexible and powerful query programs. TQL programs can perform the following functions on on-line files:

- display data
- modify data
- enter data
- generate reports

TQL accesses files indexed and direct access (relative) files as well as databases accessible through TIP/dbi.

---

## TQL Components

TQL is an integrated environment. It combines three main components into a system to provide data access services.

TQL maintains control of the system using a central control file and the operating system directory structure. It is important that all manipulation of the TQL system is done using the utilities provided for that purpose.

## Query Language

The query language is based on COBOL syntax to provide ease of programming and a small learning curve. While the language follows the syntax of COBOL it provides greater leverage when creating programs. The semantics of a number of statements go beyond the basics of COBOL.

## Development Environment

The development environment is provided by the utility TQLMON. TQLMON is used for creating all the definitions in the system and is the main utility for manipulating the TQL system.

See the [TQLMON](#) section for details on using TQLMON to create TQL applications.

## Runtime Interpreter

Once a TQL application has been created it is run by the interpreter TQLRUN. While running the program the end-user can view displays or reports programmed by the developer of the TQL application.

In addition, ad hoc queries may be executed to request specific data not covered by the provided displays or reports. These queries allow the end-user to view the data in a manner not anticipated by the programmer or to use the provided displays and reports with confinement parameters.

See the [TQLRUN](#) section for details on running TQL applications and the ad hoc command set.

---

## Porting Your Programs to TQL

This section outlines the major differences between TIP/30 TQL and TQL and the procedure for importing TIP/30 TQL applications into TQL.

### General TQL Changes

TQL no longer uses one control file for maintaining the TQL system. Source code, pseudo-code and symbol tables are all kept in UNIX directories and rely on the usual UNIX security for access. Do not access these directories directly. Always use the [TQLMON](#) utility to maintain the TQL system.

The source for programs and records is no longer kept in edit buffers. Under TIP/30 TQL, the E command or **CANCEL** out of the editor resulted in the source being compiled. TIP maintains the source separately and compilation is based upon the modification date of the source file. To compile your source, you must write out the source before exiting the editor. If using FSE as your editor, use the WZ command without any other parameters. A description of FSE can be found in ***TIP Utilities manual***.

If you end an edit session without saving the source, the posted source will not be compiled. TIP will save the edit buffer, with your changes, for the next time you update that source file.

If you will be going through multiple edit-compile cycles for a module, use the WE command to end the editor FSE. This will write and compile the source and save an edit buffer, which will load faster for the next edit session.

TQL has some default source file extensions that you may see when editing. They are as follows:

Extension	Description
<filename>xxx.trd	A TQL record definition
<filename>xxx.tpd	A TQL program definition
<filename>xxx.tfd	A TQL file definition (not associated with the TIP TFD utility)

## Language Definition Changes

Most of the changes involve the TQL language and are generally limited to program definition. Use the **cvttql** conversion program to convert your TIP/30 TQL programs to TIP. Any new programs you write must conform to the TQL program definition standards.

If you use any of the new features in TQL you may find that this will restrict porting TQL code back to OS/3. To ensure that you can move your TQL code back to OS/3, avoid the following items:

- MOVE/ADD/SUBTRACT with multiple receiving items
- MULTIPLY/DIVIDE statements
- NEXT-LOOP/BREAK statements
- Reference modification
- Subscripting with more than two subscripts
- Qualification
- Using files with more than five keys
- PUT
- SET
- Level 88 usage
- Level 66 usage
- Complex statements are delimited with { ... } pairs or BEGIN ... END pairs. The TIP/30 TQL statement delimiters ( ... ) are no longer supported.

The array range specifier ":" has been replaced with "..". This change allows you to use COBOL style reference modification. See [ANSI COBOL-85 Specifications](#) for information regarding reference modification.

Statements such as MOVE, ADD or SUBTRACT that have a GIVING or TO clause now accept multiple receiving items.

**For example:**

```
MOVE 1 TO ITEM-1 ITEM-2 ITEM-3
```

```
→will move 1 into ITEM-1, ITEM-2 and ITEM-3
```

The major side effect of this is that code may work differently under TQL. See the `cvttql` section following for details.

TIP/30 TQL treated group names used as display/report items differently depending on whether the items exist in a report or display:

When used in a report they were treated as alphanumeric (PICTURE X) items.

In displays, they meant the short form of all the non-FILLER field names in the group.

Support for this construct depends on a configuration parameter (see [UC - Update Control Record](#)) which defaults to supporting the construct. When support has been disabled, use `FIELDS/MEMBERS OF/IN <grp-name>` to achieve the same effect.

TQL is case insensitive except in two cases:

Any UNIX file names are case sensitive.

Ad hoc literals are converted to uppercase unless an "I" or "L" prefixes the literal (that is, L"Dont convert").

## cvttql - TIP/30 to TQL Converter

TQL source files from OS/3 must be run through a converter before attempting to install them into a TQL system. This is required because of changes to the TQL language that would cause some programs to either work differently or not compile.

### Syntax

```
cvttql [ -lpv ] -o directory file(s)
```

### Where:

- l** Convert literals to the correct type.
- p** Source code does not have COBOL sequence numbers (i.e. column 1 is really column 7)
- v** Verbose output. Shows the files being converted and success or failure of the conversion
- o directory**  
The directory where the converted source will be put. This must be specified. If you want to overwrite the original source specify a directory of ".".
- file(s)** The list of files to be converted.

## Required Conversions

The following is a list of the conversions made by the converter:

### Reserved word conversion.

The TQL reserved word list now includes the full set of COBOL reserved words which invalidates the names of some data items. Data names that are reserved words are converted by changing the last letter of the name to the character 'Z'. For instance, the name CLASS would become CLASZ.

### Complex statement delimiters

Complex statements were delimited by either ( ... ) pairs or BEGIN ... END. ( ... ) pairs are replaced by { ... } pairs. BEGIN ... END pairs are left alone.

### Literal conversion

If requested, literals that do not match the storage type will be converted to the correct type. Because this has the potential to change the meaning of the data this conversion is not done automatically.

### Delimiting multiple receiving items

Statements such as MOVE and ADD now accept multiple receiving items in keeping with the COBOL syntax. The following code will now work differently under TQL.

```
MOVE "A" TO FLD-1
      FLD-2 FLD-3
```

Instead of moving "A" to FLD-1 and outputting FLD-2 and FLD-3, it will attempt to move "A" to FLD-1, FLD-2 and FLD-3. The PUT verb has been introduced to handle this case.

The converter changes the preceding code to:

```
MOVE "A" TO FLD-1
      PUT FLD-2 FLD-3
```

Note that no special handling is required for the following:

```
MOVE "A" TO FLD-1
MOVE "B" TO FLD-2
```

### Data name aliases converted to level 66's

When defining a data item any data name entered as an alias is converted to a level 66.

```
05 FLD-1 FLD-ALIAS-1 FLD-ALIAS-2
      PIC XXX.
```

In the following example, FLD-1 could also be referred to as FLD-ALIAS-1 or FLD-ALIAS-2. This is no longer supported and is replaced by level 66 support. The above code is changed to:

```
05 FLD-1 PIC XXX.  
66 FLD-ALIAS-1 RENAMES FLD1.  
66 FLD-ALIAS-2 RENAMES FLD1.
```

Level 66's terminate the definition of the 01 group being defined.

#### Insert DO before loop counts

DO was optional when specifying a loop construct. The following will work differently under TQL.

```
PUT FLD-1 FLD-2 3 { FLD-3 }
```

This will output FLD-1, FLD-2, the number 3 and FLD-3 instead of putting 3 FLD-3's. This is converted to

```
PUT FLD-1 FLD-2 DO 3 { FLD-3 }
```

## Running the Converter

This section outlines the steps involved when converting TQL source. To properly convert the programs you must have access to symbol tables for the record definitions used. This is required because some conditional expressions that are followed by a complex statement groups delimited by ( ... ) pairs are ambiguous without information about any data names used in the conditional expression.

The first step is to submit all the record definitions to the converter. If the source files have been extracted off of a tape using the **armlibs** utility then the record definitions have the extension ".trd".

Once you convert the record definitions you have two choices:

- you may convert the programs immediately, or
- convert them after you have installed the records into the TQL system.

When converting programs the converter looks at the current directory first to find any required symbol tables. If it is unable to locate them, it then looks in the TQL system. If there is still no match, an error is reported and the program cannot be converted. If you install the records into the TQL system before converting the programs then the symbol tables created in the directory where the converter was run may be deleted. Symbol tables are files that have a ".sym" extension.

Submit the programs to the converter. Programs have the extension ".tpd" if extracted from tape via **armlibs**.

When all the source has been converted you may use the TQLMON commands C and CP to install the source into the TQL system. See the TQLMON section for details.

The symbol tables in the directory where the conversion was performed may now be deleted.



**Example:**

```
cvttql -v -o ./ *.trd
*> convert all records overwriting the
*> original source with the converted
*> source.
cvttql -v -o cvtd *.tpd
*> convert all programs putting the
*> converted source into the directory
*> cvtd
```

## Importing TQL Applications into TQL

This section describes the recommended procedure for importing TQL applications into the TQL system. This assumes that the source has been processed by the converter **cvttql**.

The following steps illustrate importing two provided TQL applications, TQLSVE and TQLTSP. TQLSVE is the TQLRUN Saved Command file maintenance program and TQLTSP is the TQL version of TSP (TIP Sample Program) which shows some basic uses of TQL.

Details on the AF, C, and CP commands mentioned may be found in the [TQLMON](#) Commands section.

**Step 1:**

Start TQLMON

```
TIP?►TQLMON
```

**Step 2:**

Define the files to TQL.

```
AF TQLSVE
AF TSPFILE
```

**Step 3:**

Define the records to TQL.

```
C SVEREC.trd
C TQLTSPR.trd
```

**Step 4:**

Define the program to TQL

```
CP TQLSVE.tpd
CP TQLTSP.tpd
```

**Step 5:**

Exit from TQLMON. Import the required screen formats to TIP using the **msgar** utility. This step is usually not required because the screen formats are already available to TIP.

At this point the TQL programs TQLSVE and TQLTSP are installed in the TQL system and available for execution.

## Saved Command File Conversion

This section outlines the step required to move the saved command file TQL\$SVE from OS/3 to UNIX. A conversion is necessary because TQLRUN has a larger command line and therefore the record size for the saved commands must be adjusted.

The structure of the file is the same except that the record length for the TQL saved command file is 486 instead of 256.

Create a 9-track tape using the OS/3 data utilities. The input record length is 256 and the output record length is 486. The saved command file is called TQL\$SVE.

Use fcsload to create the UNIX file tq1\$sve. The copybook to supply to fcsload is \$TIPROOT/src/SVEREC.cpy. This copybook is a modified version of the TQL saved command file record definition SVEREC.trd.

See the fcsload documentation for details.

## TQLCC - TQL Compiler

This chapter describes the rules of syntax for the specification of FILES, RECORDS, and PROGRAMS in the TIP Query Language.

---

### Reserved Words

The table that follows contains words that have reserved meaning to both TQL and COBOL-85. The reserved words may not be used outside of the reserved meaning in a TQL program.

#### TQL RESERVED WORDS

ACCEPT, ACCESS, ADD, ADVANCING, AFTER, ALL, ALLOW, ALLOWED, ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, ALPHANUMERIC, ALPHANUMERIC-EDITED, ALSO, ALTER, AND, ANY, APPLY, ARE, AREA, AREAS, ASC. ASCENDING, ASSIGN, AT, AUTHOR, AUTHOR\$

BEFORE, BEGIN, BEGINS, BINARY, BLANK, BLOCK, BOTTOM, BREAK, BY

CALC, CALL, CANCEL, CASE, CASEOF, CD, CF, CH, CHANGE, CHANGED, CHARACTER, CHARACTERS, CLASS, CLOCK-UNITS, CLOSE, CODE, CODE-SET, COLLATING, COLUMN, COMMA, COMMIT, COMMON, COMMUNICATION, COMP, COMP-1, COMP-2, COMP-3, COMP-4, COMP-X, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-4, COMPUTATIONAL-X, COMPUTE, CONFIGURATION, CONTAIN, CONTAINS, CONTENT, CONTINUE, CONTROL, CONTROLS, CONVERTING, COPY, CORR, CORRESPONDING, COUNT, CURRENCY, CURRENT

DATA, DATE, DATE-COMPILED, DATE-WRITTEN, DAY, DAY-OF-WEEK, DD\$, DE, DEBUG, DEBUG-CONTENTS, DEBUG-ITEM, DEBUG-LINE, DEBUG-NAME, DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3, DEBUGGING, DECIMAL-POINT, DECLARATIVE, DECLARATIVES, DEFAULT, DELETE, DELIMITED, DELIMITER, DEPENDING, DESC, DESC\$, DESCENDING, DESTINATION, DETAIL, DIRECT, DISABLE, DISPLAY, DISPLAY1, DIVIDE, DIVISION, DMS, DMY\$, DO, DOES, DOWN, DROP, DUPLICATES, DYNAMIC

EDIT\$, EGI, ELSE, EMI, EMPTY, ENABLE, END, END-ACCEPT, END-ADD, END-CALL, END-COMPUTE, END-DELETE, END-DISPLAY, END-DIVIDE, END-EVALUATE, END-IF, END-MULTIPLY, END-OF-PAGE, END-ON, END-PERFORM, END-READ, END-RECEIVE, END-RETURN, END-REWRITE, END-SEARCH, END-START, END-STRING, END-

SUBTRACT, END-UNSTRING, END-WRITE, ENTER, ENTRY, ENVIRONMENT, EOP, EQ, EQUAL, EQUALS, ERRCODE\$, ERROR, ESI, EVALUATE, EVERY, EXCEPTION, EXECUTE, EXHIBIT, EXIT, EXPORT, EXTEND, EXTERNAL

FALSE, FD, FETCH, FIELDS, FILE, FILE-CONTROL, FILE-SERVER, FILLER, FINAL, FIRST, FOOTING, FOR, FROM

GE, GENERATE, GET, GIVING, GLOBAL, GO, GOTO, GREATER, GROUP, GROUPS, GT

HEADING, HELP, HH\$, HHMM\$, HIDDEN, HIGH-VALUE, HIGH-VALUES, HOME\$

I-O, I-O-CONTROL, ID, IDENTIFICATION, IDENTIFIER, IF, IFDMS, IN, INDEX, INDEXED, INDICATE, INITIAL, INITIALIZE, INITIATE, INPUT, INPUT-OUTPUT, INSERT, INSPECT, INSTALLATION, INTO, INVALID, INVOKE, IS

JUL\$, JUST, JUSTIFIED

KEY

LABEL, LAST, LE, LEADING, LEFT, LESS, LEVEL\$, LIMIT, LIMITS, LINAGE, LINAGE-COUNTER, LINE, LINE\$, LINE-COUNTER, LINES, LINKAGE, LIST, LOCATION, LOCK, LOW-VALUE, LOW-VALUES, LPP\$, LT

MAX, MAXREAD, MEMBER, MEMBERS, MEMORY, MERGE, MESSAGE, MIN, MIN\$, MODE, MODIFIED, MODIFY, MODULES, MON\$, MORE\$, MOVE, MULTIPLE, MULTIPLY, MUST

NAMED, NATIVE, NE, NEGATIVE, NEXT, NEXT-LOOP, NL\$, NO, NOT, NULL, NUMBER, NUMERIC, NUMERIC-EDITED

OBJECT-COMPUTER, OCCURS, OF, OFF, OMITTED, ON, ONLY, OPEN, OPTIONAL, OR, ORDER, ORGANIZATION, OTHER, OUTPUT, OVERFLOW, OWNER

PACKED-DECIMAL, PADDING, PAGE, PAGE\$, PAGE-COUNTER, PARAGRAPH, PASSWORD, PERFORM, PF, PH, PIC, PICTURE, PLUS, POINT, POINTER, POSITION, POSITIVE, PREFIX, PREV, PRINT, PRINTING, PRIOR, PRIORITY, PROCEDURE, PROCEDURES, PROCEED, PROGRAM, PROGRAM\$, PROGRAM-ID, PROTECT, PURGE, PUT

QUEUE, QUOTE, QUOTES

RANDOM, RANGE, RD, READ, READY, RECALL, RECEIVE, RECORD, RECORDS, REDEFINES, REEL, REFERENCE, REFERENCES, RELATIVE, RELEASE, REMAINDER, REMOVAL, REMOVE, RENAMES, REPLACE, REPLACING, REPORT, REPORTING, REPORTS, RERUN, RESERVE, RESET, RETURN, REVERSED, REWIND, REWRITE, RF, RH, RIGHT, ROUNDED, RUN

SAME, SAVE, SD, SEARCH, SECTION, SECURITY,  
 SEGMENT, SEGMENT-LIMIT, SELECT, SELECTIVE, SEND,  
 SENTENCE, SEPARATE, SEQUENCE, SEQUENTIAL, SET,  
 SHOW, SIGN, SITE\$, SIZE, SKIP\$, SORT, SORTED, SORT-  
 FILE-SIZE, SORT-MERGE, SORT-MODE-SIZE, SOURCE,  
 SOURCE-COMPUTER, SPACE, SPACES, SPECIAL-NAMES,  
 SPECIFIC, STANDARD, STANDARD-1, STANDARD-2,  
 START, STATUS, STOP, STORE, STRING, STRUCTURE,  
 SUB-QUEUE-1, SUB-QUEUE-2, SUB-QUEUE-3, SUBTRACT,  
 SUM, SUPPRESS, SYMBOLIC, SYNC, SYNCHRONIZED,  
 SYSTEM

TAB\$, TABLE, TALLYING, TAPE, TERMINAL, TERMINATE,  
 TEST, TEXT, THAN, THEN, THROUGH, THRU, TID\$, TIME,  
 TIME\$, TIMES, TO, TOP, TRACE, TRAILING, TRANSFORM,  
 TRUE, TYPE

UID\$, UNIT, UNSTRING, UNTIL, UP, UPDATE, UPON,  
 USAGE, USE, USING

VALUE, VALUES, VARYING, VERIFY, VIA

WHEN, WHEN-COMPILED, WHERE, WHILE, WITH, WORDS,  
 WORKING-STORAGE, WRITE

YMD\$, YY\$

ZERO, ZEROES, ZEROS

---

## COPY Statement

The COPY statement allows you to specify the name of the file from which the compiler reads the TQL source when the COPY statement is executed.

### Syntax:

```
COPY text-name|external-filename  

[OF|IN library-name|library-name-literal]
```

### Where:

**text-name**

A unique external file name which cannot contain a "/" character or possess an extension.

**external-filename**

A UNIX file name which must be contained by quotes (" ") and may contain a "/" character or possess an extension.

**library-name**

may be one of:

the name of an environment variable containing a directory name in which the copybook resides. The variable can also be defined as "DD\_library-name".

a Directory in the current working Directory.

#### library-name-literal

A UNIX Directory PATH. Must be in quotes and may contain a "/" character and extension.

Note: The text-name may also reside in a directory specified in the environment variable COBCPY. COBCPY is a colon-delimited list of directories.

The order of qualification for these possibilities is:

If no library is specified: text-name, text-name.cbl, text-name.cpy

If a library is specified: library/text-name, library/text-name.cbl, library/text-name.cpy

If a library is specified and an environment variable named library is set, for example, \$library/text, etc. where \$library is the contents of the environment variable.

If a library is specified and an environment variable named library is set, for example, DD\_library/text, etc. where DD\_library is the contents of the environment variable.

Each directory in \$COBCPY (if set) is searched.

---

## Fields

Fields may be defined in record layouts or in the (optional) working-storage section of a TQL program. Fields are defined in a manner similar to standard COBOL data division items.

Some special considerations exist for TQL programs:

- fields may be subscripted by enclosing the subscript value within parentheses.
- an array may be defined to have up to seven dimensions (subscripts).
- a subscript may be a literal value, an expression, or another field name.

TQL also supports COBOL 85 reference modification and qualification. See [ANSI COBOL-85 Specifications](#) for a discussion of these concepts.

As a coding convenience, when a field name is used to implicitly output the field in a pre-defined display or report, a range of items can be implied by using a special notation:

**Example:**

```

05  AN-ARRAY           OCCURS 10 TIMES.
    10  AN-ITEM        PIC X(8) .
    10  ANOTHER-ITEM   PIC X(8) .
    
```

The first 3 items could be output in a display or a report simply by using the appropriate subscripts:

```

AN-ITEM (1) AN-ITEM (2) AN-ITEM (3)
    
```

Or, alternatively and more compactly, using a special range notation:

```

AN-ITEM (1..3)
    
```

**Additional Considerations:**

You may use qualified field names in TQL, for example:

```

05  FOO.
    10  FOO-A.
05  FOO1.
    10  FOO-A.
...
...
FOO-A OF FOO1 *> references FOO-A OF FOO1
    
```

TQL supports **Scaled Numeric Fields** placing a **P** (or place holder) in PICTURE clauses. The numbers are processed and displayed as if the **P** was an actual digit with a zero value. Whenever a value is stored into these fields the number is adjusted so that the **P** digits are not stored in the record. This is also how COBOL treats this number.

**Example :**

```

05  DOLLAR             PICTURE S99999PPP.
05  SMALLNUM          PICTURE VPPP999.
    
```

TQL will accept any unambiguous abbreviation wherever you specify a data field name. TQL first searches for an exact match on the given field name. If the match fails TQL performs a complete search of the symbol table looking for the supplied name as any substring of a symbol. If only one possible match is found the field name is known. When two or more possible matches are found, TQL will display the following error message:

## Ambiguous Field References

During the execution of a TQL program, TQL will also display all names that match the search.

In the sample data file and sample TQL program the data field name CM-DP-MGR may be referenced as simply MGR, CM-MACHINE may be referenced as MACH or MACHINE.

This technique allows data processing to use data processing data field names while an end user may omit prefixes and suffixes to enter more meaningful data field names.

TQL does not display fields labelled as FILLER. See the syntax of **field** in the [DISPLAY DIVISION](#) section of this manual.

TQL supports level 88 items (also known as condition names.) It follows COBOL-85 syntax with COBOL-99 extensions.

The declaration syntax is:

```

88  condition-name-1
    { VALUE [IS]|VALUES [ARE]} literal-1
    [{ THRU|THROUGH } literal-2]...
    [ WHEN [ SET TO ] FALSE literal-3].

```

When the 88 level is SET to FALSE it becomes literal-3.

Each literal must be a compatible value for condition-name-1.

A condition name may be moved to any compatible data item as long as the condition name has only one value associated with it.

Condition names may not be reference-modified.

The verb SET in the TQL syntax provides for level 88 support. A condition name can also be set to FALSE if a **when false** literal is specified:

**Example:**

```

05  FOO                PICTURE X(4) .
05  FOO-TOO           PICTURE X(4) .
    88  FOO-88-1       VALUE "ABCD" "DEFG" .
    88  FOO-88-2       VALUE "TEST" .
    88  FOO-88-3       VALUE "ABCD"
                       WHEN FALSE "DCBA" .

MOVE FOO-88-2         TO FOO ①
MOVE FOO-88-1         TO FOO ②
MOVE FOO-88-3         TO FOO ③

SET FOO-88-2          TO TRUE ④
SET FOO-88-1          TO TRUE ⑤
SET FOO-88-3          TO FALSE ⑥
SET FOO-88-2          TO FALSE ⑦

```

**Notes:**

- ① Moves "TEST" into FOO.



- ② Not valid, multiple values in condition name.
- ③ Not valid, multiple values.
- ④ Moves "TEST" into FOO-TOO.
- ⑤ Invalid multiple values.
- ⑥ Moves "DCBA" into FOO-TOO.
- ⑦ Invalid no FALSE literal.

---

## TQL Expressions

TQL allows the programmer or run-time user to make use of arithmetic and relational expressions. These expressions may be used either as part of the TQL program proper or as part of a run-time command (example: in the run-time "IF" command).

This section describes the syntax of the general TQL expression and contains several example expressions.

### Syntax:

```

field
value
expr OPER expr
expr CONNECTOR expr
NOT expr
(expr)

```

### Where:

- ( ) The use of parentheses may be necessary to force a specific order of evaluation of the expression or to nest expressions.

If parentheses are not used, standard operator precedence rules apply (multiplication and division before addition and subtraction etc.).

**field** The name of a field that is defined in the TQL program.

A list of available field names can be found (at run-time) by using the "SHOW" command (documented in a following section).

**value** A numeric or character value (literal).

Numeric literals must be entered without any comma separator characters.

Numeric literals may specify a leading or trailing sign and a decimal place (if required).

Numeric values less than 1 must be entered with a leading zero before the decimal; example: 0.35 not .35

## OPER

A relational or arithmetic operator (arithmetic operators may only be applied to numeric fields!).

TQL supports the operators listed in the following table.

Operator	Alias	Description
EQ	=	equal
NE	< >	not equal
GT	>	greater than
LT	<	less than
GE	> =	greater than or equal
LE	< =	less than or equal
BEGINS WITH	= *	begins with
DOES NOT BEGIN WITH	= !	does not begin with
CONTAINS	= :	contains
DOES NOT CONTAIN		does not contain
+		arithmetic addition
-		arithmetic subtraction
*		arithmetic multiplication
/		arithmetic division
%		arithmetic remainder

The connectors support by TQL are as follows:

Connector	Alias	Description
AND	&	logical "and" function
OR		logical "or" function (meaning either or both)
NOT		"NOT" logical negative.

### Example:

```
MOVE INVENTORY-COUNT - 1 TO WORK-COUNT.
IF (JOB-DESCRIPTION CONTAINS 'DEPUTY') AND SALARY > 25000
AND SALARY < = 50000
```

```
IF (NOT JOB-DESCRIPTION =: 'DEPUTY')
IF 0 = TOTAL-COUNT % 2
```

The field name "SALARY" had to be repeated for the comparison with 50000. This illustrates that TQL does not allow the subject of a comparison to be omitted, as COBOL-85 does.

Additional Considerations:

Numeric values must be specified without comma separators. (example: 25000 rather than 25,000 ).

The remainder operator % involves an implied division; the result is the remainder rather than the quotient.

The last example compares 0 with the remainder when TOTAL-COUNT is divided by 2. If the remainder is zero, it implies that the field was evenly divisible by 2.

The result of a relational operator (example: A >= B) is considered to be (numeric) 1 if the result is TRUE and a (numeric) 0 if the result is FALSE. Such implied numeric values may be used in further computations if desired.

---

## File Definition

The programmer must define all required on-line files to TQL using the TIP TCM program SMFILE and the TQLMON AF or NF command. To define currently existing host computer TIP/30 TQL files, you may wish to create a source module containing information from the TIP/30 generation parameters for the files. Alternately, the TQLMON command NF can be used to manually enter the information for each file.

The source module may contain one or more FILE definitions. This source module is then compiled by TQL (the compilation process will be described in detail in a later section).

The syntax requirements are:

### Syntax:

```
FILE lfn, filetype
[ ACCESS= ]
[ BLKSIZE= ]
[ DELETE= ]
[ INDSIZE= ]
KEYLEN=
KEYLOC=
KEY1=
...
KEY10=
```

[ RECFORM= ]  
RECSIZE=

.

Entries marked with [ ] are not required by TQL, and may be omitted. A required period ends the definition of each file.

**Where:**

Lfn

The Logical file name of the file (as defined in the TIP catalogue).

Filetype

The type of file.

Choose one of ISAM, or DAM.

ACCESS=

Not required; ignored.

BLKSIZE=

Not required; ignored.

DELETE=

Not required; ignored.

INDSIZE=

Not required; ignored.

KEYLEN=

The length of the key for the file.

This keyword is normally used only when the file in question has a single key.

KEYLOC=

The zero relative location of the key in the record. This value is the number of bytes that precede the key.

Default=0.

KEYLEN and KEYLOC do not have to be specified if the key information is provided by one or more of the keywords KEY1= through KEY10=.

KEY1=(size,loc,NDUP,NCHG)

This keyword defines index 1.

The value "size" is the key length in bytes.

The value "loc" is the zero relative key location (the number of bytes that precede the key).

**Note:** TIP does not allow the primary key of a MIRAM file to change or have duplicates.

KEY10=(size,loc,NDUP|DUP,CHG|NCHG)

This keyword defines index 10. A file can have up to ten keys.

RECFORM=

Record format.

Choose either FIXBLK (fixed block) or VARBLK (variable blocked). Default=FIXBLK.

**Note:** The first halfword of a variable length record is the record length. This record length field is accessible by the TQL program and the record definition must account for these two bytes (a binary halfword).

When TQL is adding a variable length record, the record length is set to the maximum record length

RECSIZE=n

The length of the records in the file.

The end of each file definition must be marked by a period following the last keyword specification.

Other file definitions may follow in the same source module (if desired).

#### **Additional Considerations:**

Much of the information described above is not strictly necessary since TQL uses the TIP file system (TIPFCS) to perform I/O to files. The KEY information is critical however, to ensure that TQL is correctly specifying key information when accessing files.

Using the NF and AF commands will ensure that the TQL file definition is the same as the TIP definition.

---

## **Record Definition**

The programmer has several methods of handling record layouts:

pre-compile the record definition and reference it by name in TQL programs that need to access such records;

Use the COPY clause to include the record layout in the TQL programs that access the record;

explicitly code the record definition in the TQL programs that access the record.

The first method (pre-compilation) is the most efficient and is highly recommended. Use of the COPY clause is clearly better than explicitly coding the record layout. The latter two methods are inferior. Pre-

compilation ensures that all TQL programs use the same record layout and reduces program compilation overhead.

The record definition follows standard COBOL record description conventions with the following exceptions:

VALUE clauses are ignored.

COMPUTATIONAL-1 and COMPUTATIONAL-2 fields (short and long format floating point) are not supported by TQL.

Level 77 items are ignored.

If the record layout is to be pre-compiled, the first statement of the source must be the FOR clause (to be described) to associate the record layout with an already defined FILE.

If the record layout is to be pre-compiled, the RECORD clause may be specified to name the record. If no RECORD clause is specified the record will be given the same name as the 01 level name. This 01 level name will be truncated to eight characters.

In the second method, the record layout is included via a COPY clause or is explicitly coded in the TQL program, it cannot have a FOR clause or a RECORD clause.

Regardless of how the record definition is coded, the record layout may contain and/or be followed by ALLOW CHANGE clauses, VERIFY clauses, ID clauses, ALLOW DELETE clauses, ALLOW ADD clauses (to be described in a following section).

Records are not allowed to change or be deleted at runtime unless such permission is explicitly granted through the appropriate clauses.

**Example:**

```
[ DECIMAL-POINT IS COMMA. ]  
[ CURRENCY SIGN IS "$". ]  
FOR PAYFILE.  
[ RECORD PAYMST. ]  
01 PAYMST.  
05 KEY.  
10 DEPT PIC 99.  
10 NUMB PIC 9(5) COMP-3.  
05 NAME PIC X(20).  
05 ADDRESS.  
10 LINE-1 PIC X(20).  
10 LINE-2 PIC X(20).  
05 SALARY PIC 9(4)V99.  
ID IS DEPT > 0.  
ALLOW CHANGE ALL.  
NO CHANGE DEPT NUMB.  
VERIFY SALARY 6000 THRU 32000.
```

**Additional Considerations:**

The definition of the key field is critical to the run-time operation of TQL. The first definition of the key field(s) is taken as the way the key will be entered by the run-time user when selecting records.

This can be a problem if the key is actually made up of several smaller fields. For example, if the key is defined as three (3) small fields then any key value must be entered at execution time as three separate items of the correct type.

Numeric data is entered as a number, but alphanumeric data must be entered in single quotes. If you prefer to enter the key data as one big field but still want to reference the sub-fields then code the record layout with one single field and then redefine it as the sub-fields. Since the single field definition would appear first, TQL would expect the key to be entered as a single data item at run time.

The specification of the currency sign and decimal point is comma must appear before the FOR *file* statement.

---

## Group Items and TQL

Group level items which are defined in a TQL record layout (or in the working-storage section) are treated by TQL in the following manner:

when a group item is used in a DISPLAY, MOVE, IF, REPORT, LIST or EXPORT command, TQL will interpret the group name as if it was a picture X field of the size of the group.

To use group-name as a short form for its fields (without including FILLERS) use:

```
FIELDS | MEMBERS OF | IN group-name
```

**Example:**

```
05 PART-NO .  
10 BRANCH PIC X(3) .  
10 PART-ID PIC X(8) .
```

The group name PART-NO is treated as a PIC X(11) field if it is moved to another field or is used in an EXPORT command.

---

## ALLOW: Changing Fields

Records are not allowed to be added, changed or deleted unless explicit permission is stated in the TQL program. Fields within records cannot change unless permission is explicitly given.

The ALLOW clause enables the programmer to specify what actions are permitted. The ALLOW clause may appear within a pre-compiled record layout, or within the [DATA DIVISION](#) of the TQL program. The program may specify multiple ALLOW clauses for a record.

**Syntax:**

```
ALLOW ADD .
ALLOW DELETE .
ALLOW CHANGE field-names .
ALLOW CHANGE ALL .
NO CHANGE field-names .
NO CHANGE ALL .
```

**Where:****ALLOW ADD**

Indicates that records may be added to this file.

**ALLOW DELETE**

Indicates that records may be deleted from the file.

**ALLOW CHANGE**

Defines which fields of the record may be changed when records are being updated.

**field-names**

A list of field names involved. The names may be separated by commas or spaces and the statement should be terminated with a period.

The reserved word ALL may be used as a field name to avoid having to explicitly mention all field names.

If the field name is a table the change attribute applies to every occurrence in the table. This field name may be subscripted to restrict application of the attribute.

**NO CHANGE**

Identifies fields, which may not change.

**Example:**

```
ALLOW CHANGE ALL .
NO CHANGE S-I-N .
ALLOW CHANGE SALARY DEDUCTIONS .
```

**Additional Considerations:**

Multiple clauses may be specified. In fact, some of these clauses may be specified within a pre-compiled record layout and then additional clauses may be specified in the program after the pre-compiled record layout is selected.

The rule used to resolve multiple specifications (for a specific field) is that the last stated attribute applies. In the example above, the field S-I-N



cannot change but all others can change (the net effect of stating: ALLOW CHANGE ALL. NO CHANGE S-I-N).

---

## ALLOW: Exporting Fields

Field names may be used by the run-time user to EXPORT data to an external destination. The programmer may specify whether or not certain fields may be EXPORTed by the user.

The default is that all fields are exportable.

The ALLOW clause may appear within a pre-compiled record layout, or within the DATA DIVISION of the TQL program. The program may specify multiple ALLOW clauses for a record.

### Syntax:

```
ALLOW EXPORT field-names.  
ALLOW EXPORT ALL.
```

```
NO EXPORT field-names.  
NO EXPORT ALL.
```

### Where:

#### ALLOW EXPORT

Defines which fields of the record may be exported.

#### field-names

A list of field names involved. The names may be separated by commas or spaces and the statement should be terminated with a period.

The reserved word ALL may be used as a field name to avoid having to explicitly mention all field names.

If the field name is a table the export attribute applies to every occurrence in the table. This field name may be subscripted to restrict application of the attribute.

NO EXPORT Identifies fields, which may not be exported.

### Example:

```
NO EXPORT ALL.  
ALLOW EXPORT CUST-NO CURRENT-BALANCE.
```

This example illustrates preventing the EXPORT of all fields except the CUST-NO and CURRENT-BALANCE fields.

### Additional Considerations:

Multiple clauses may be specified. In fact, some of these clauses may be specified within a pre-compiled record layout and then additional clauses

may be specified in the program after the pre-compiled record layout is selected.

The rule used to resolve multiple specifications (for a specific field) is that the last stated attribute applies.

---

## ALLOW NULL: Fields

The NULL attribute may be applied to a field in a record that is stored in a TIP/dbi-supported database. TIP/dbi represents a NULL field by using LOW-VALUES for the contents of the data field.

Displaying and updating such fields using TQL requires special handling. The LOW-VALUES in the data field could cause incorrect data to be displayed on a screen format because the LOW-VALUES have special meaning in the context of a TIPMSGO call.

To ensure that data for a record is displayed and updated correctly TQL will convert any display field that contains all LOW-VALUES to all SPACES or ZEROES. If the record was being updated using a display, SPACES or ZEROES could be stored back in the record instead of the LOW-VALUES that the field originally contained.

To address this situation, fields can be given the NULL attribute, which will cause the following behavior:

If, when updating a record using a TQL display, a field originally contained LOW-VALUES and after input is received it contains all SPACES for non-numeric or ZEROES for numeric then TQL will automatically store LOW-VALUES in the field. If the NULL attribute is not specified, SPACES or ZEROES will be stored in the field.

The original value of the field must have been LOW-VALUES for TQL to exhibit this behavior. When adding a new record using a TQL display the record will be initialized to SPACES for non-numeric and ZEROES for numeric. If you want fields to be stored as NULL when adding a new record then LOW-VALUES should be moved to the desired fields in an ON ADD/WRITE declarative. TQL will not automatically handle this scenario.

### Syntax:

```
ALLOW NULL field-names .
```

### Where:

#### **ALLOW NULL**

Identifies fields of the record, which must be given an explicit value when a record is added or changed.

#### **field-names**

A single field name or a list of field names.

The names may be separated by commas or spaces and the statement must be terminated with a period.  
If the field name is a table the NULL attribute applies to every occurrence in the table.

**Example:**

```
ALLOW NULL SALARY, DEDUCTIONS.
```

**Additional Considerations:**

Multiple ALLOW NULL clauses may be specified if necessary.

---

## ALLOW: GO (Auto Update)

TQL has a run-time verb (GO) which may be specified in conjunction with the UPDATE command. This option instructs TQL to update all records selected without displaying each record on the terminal.

This facility is permitted at run time only if an ALLOW GO clause appears in the TQL program.

This clause can not appear in a pre-compiled RECORD definition. It must be coded in the program! This is a safety feature; precompiled record layouts (in our view) should not freely provide such potentially dangerous permission.

**Syntax:**

```
ALLOW GO.
```

**Example:**

An example illustrating the use of the GO verb is included in the description of the [UPDATE](#) run-time command.

---

## Hidden Fields

The run-time user has access to a SHOW command which is used to reveal record and field names the user often cannot remember all of the field names in a particular record or display.

The programmer may identify which fields are to be considered to be hidden fields. Hidden field names will not be revealed by the [SHOW](#) command, but are otherwise valid field names (the user may use the field name if it is known or guessed, but the SHOW command will not reveal the existence of such fields).

The default is that all fields are not hidden.

This clause may appear within a pre-compiled record layout, or within the [DATA DIVISION](#) of the TQL program. The program may specify multiple such clauses for a record.

**Syntax:**

```
HIDDEN field-names.  
HIDDEN ALL.  
NO HIDDEN field-names.  
NO HIDDEN ALL.
```

**Where:****HIDDEN**

Defines which fields of the record are to be hidden from the run-time user.

**field-names**

A list of field names involved. The names may be separated by commas or spaces and the statement should be terminated with a period.

The reserved word ALL may be used as a field name to avoid having to explicitly enter all field names.

**NO HIDDEN**

Identifies fields, which are not hidden.

**Example:**

```
HIDDEN ALL.  
NO HIDDEN CUST-NO CURRENT-BALANCE.
```

In this example the SHOW command may only reveal the existence of the fields named CUST-NO and CURRENT-BALANCE.

**Additional Considerations:**

Multiple clauses may be specified. In fact, some of these clauses may be specified within a pre-compiled record layout and then additional clauses may be specified in the program after the pre-compiled record layout is selected.

The rule used to resolve multiple specifications (for a specific field) is that the last stated attribute applies.

---

## MUST ADD: Fields

If there are fields, which must be entered when the run-time user is entering data at the terminal, the programmer must designate such fields as "MUST ADD" fields.

Numeric fields designated as **MUST ADD** are not accepted from the terminal if the value is omitted or zero. Alphanumeric fields are not accepted if the field is omitted or contains all spaces.

The programmer may specify the following statements after the record definition. The default for a field is that TQL will accept fields as entered (subject to any [VERIFY](#) clauses that may be present for the field).

**Syntax:**

```
MUST ADD field-names .  
MUST ADD ALL .
```

**Where:****MUST ADD**

Identifies fields of the record, which must be given an explicit value when a record is added or changed.

**field-names**

A single field name or a list of field names.

The names may be separated by commas or spaces and the statement should be terminated with a period.

If the field name is a table the change attribute applies to every occurrence in the table. This field name may be subscripted to restrict application of the attribute.

**ALL** The reserved word "ALL" may be used to imply all that all fields in the record are affected.

**Example:**

```
MUST ADD ALL .  
MUST ADD SALARY , DEDUCTIONS .
```

**Additional Considerations:**

Multiple **MUST ADD** clauses may be specified if necessary.

---

## Preventing Use of Run-time Commands

When a TQL program is executed by a user, the user normally has access to a number of run-time commands. The TQL programmer can prevent the use of some run-time commands by including one or more of the following statements at the end of the [DATA DIVISION](#) coding.

**Syntax:**

```
NO PRINT  
NO SELECT  
NO SORT
```

In each case, the presence of the statement prohibits the run-time user from using the corresponding statement while executing the TQL program.

---

## Record Selection - ID IS

Files may contain many different record types. When more than one [RECORD](#) type exists in a particular file, TQL must have a means of determining which record layout applies to a given physical record.

The ID clause is used to provide the criteria for TQL to determine whether a specific record is of a particular layout or type.

### Syntax:

```
    ID IS expression
```

### Where:

#### expression

A relational expression which is a test for the presence of this record.

TQL evaluates the expression on every read or write of a record to determine whether the physical record is to be described by this record layout.

### Example:

```
    ID IS REC-TYPE = 'H' .  
    ID IS REC-TYPE NE 'H' AND SAL > 25000 .
```

In the first example, the programmer has specified that the field REC-TYPE must be equal to the literal "H".

The second example requires that the field REC-TYPE is NOT equal to the literal "H" and the field SAL must be greater than 25,000.

If a record is read that does not satisfy the condition, TQL ignores that record and proceeds to the next record.

It is not possible to use Working Storage fields or System fields in expressions in an ID IS clause.

---

## Key Prefix

A popular technique to segregate records in a file is to specify a sequence of characters as a prefix within the key of a record. For example, a payroll file may contain records for many client companies.

The key is constructed by concatenating a particular company code (say one character) to an employee number:

```

05  PAY-KEY.
    10  COMP-CODE          PIC X.
    10  EMP-LAST-NAME     PIC X(20) .
    
```

When this type of organization is used, the intention is often to hide from the run-time user the existence of the prefix that appears at the front of all key values. TQL can be made aware of the existence of a common prefix and made to automatically prefix all supplied keys with the stated alphanumeric value.

#### Syntax:

```

KEY PREFIX [ FOR { data-name | KEYn } ... ]
Literal
    
```

#### Where:

##### data-name

Restricts KEY PREFIX to the specified key. The key may be specified using the data name corresponding to the key.

If the FOR clause is omitted the key prefix applies to all keys.

##### KEYn

Restricts KEY PREFIX to the specified key. The key may be specified using the KEYn keyword where n is a number in the range of 1 to 10 (i.e. KEY5).

If the FOR clause is omitted the key prefix applies to all keys.

##### literal

A character literal representing the common character prefix that is to be automatically used as a prefix to key values entered by the run-time user.

#### Example:

If (referring to the key layout described at the beginning of this section) the COMP-CODE field always contained "J" for the users of this particular TQL program, the clause:

```
KEY PREFIX 'J' *> Use literal "J"
```

could be included following the record definition.

At run time, the terminal user would have the impression that the key of the file was simply the EMP-LAST-NAME information and could enter commands that omit the "J" that is always present:

```

REC FROM 'SMITH'
*> using key prefix example from above
    
```

**\*> implies "JSMITH" is the real key**

#### **Additional Considerations:**

The KEY PREFIX applies to all keys of a MIRAM file, not just the primary key if the FOR clause is not specified. This clause cannot be specified in a pre-compiled record layout. It must appear in the program (of course, it can simply be placed after the reference to a pre-compiled record).

---

## **VERIFY: Fields**

The programmer may specify that TQL is to verify the contents of one or more fields whenever a record is added or updated.

TQL validates the contents of the fields as the data fields are moved from the display area to the record (build) area. Field verification is only applied to fields displayed to a user. Verify clauses for fields that are not displayed to the user are ignored.

Fields may be verified by specifying a list of possible values for each field to be verified. Such statements must follow the appropriate record definition.

If the field name is an unsubscripted table item, the VERIFY literals apply to every occurrence of the table. You may specify separate verifies for individual table items by specifying subscripts as required for the field name.

VERIFY clauses are accumulative. Previous VERIFYs continue to apply and the new literals are added as required.

#### **Syntax:**

```

VERIFY field 'string' THRU 'string'.
VERIFY field 'string', 'string' ... 'string'
VERIFY field 'string'.
VERIFY field number THRU number.
VERIFY field number, number ... number.
VERIFY field number.

```

#### **Where:**

**field**

The name of the data field to be verified.

**'string'**

A specific alphanumeric value.

**number**

A specific numeric value.



**THRU or THROUGH**

Used to define an inclusive range of values.

**Example:**

```
VERIFY SALARY 10000, 20000, 30000 THRU 55000.
VERIFY TITLE 'V.P.', 'MANAGER', 'GO-FOR'.
VERIFY TABLE-1 100 *> verify applies to every occurrence
VERIFY TABLE-2 (1..5) 100 *> verify applies to first 5 occurrences
VERIFY TABLE-2 (1..2) 200 *> in addition, TABLE-2(1..2) may also
*>have the value 200 as well as 100.
```

**Additional Considerations:**

A field may be tested for specific values and/or range(s) of values. If the value of a field (that is entered at run time by the terminal operator) is found to not pass all specified VERIFY clauses, TQL sends an error message to the terminal operator. The terminal operator must correct the field in error and press the **TRANSMIT** key to try again.

VERIFY clauses do not apply to TQL statements used to modify fields. So a programmer can MOVE any value to a field regardless of whether there are VERIFY statements for the field.

If a VERIFY clause exists for a field then a MUST ADD statement for the field is not necessary and can lead to confusion. To allow an alphanumeric field to be left blank include ' ' as a valid value for the field. To allow a numeric field to be left blank include 0 as a valid value for the field. However, if MUST ADD was specified for the field then a value must be entered for the field even if the VERIFY clause specified that blank or zero was a valid value. This same behavior can be achieved by omitting the MUST ADD for the field and simply removing blank or zero from the list of valid field values.

The following are two ways of imposing identical restrictions on FIELD-A.

```
MUST ADD FIELD-A
VERIFY FIELD-A ' ' 'A' 'B' 'C'
VERIFY FIELD-A 'A' 'B' 'C'
```

This VERIFY clause would not prevent the programmer from moving some other value to FIELD-A. For example:

```
ON WRITE OF RECNAME
MOVE 'Z' TO FIELD-A.
```

---

**System Fields**

There are several system data fields that are maintained by TQL that are available to the TQL program. They may be used in the same manner as record fields with the exception that ONLY the ERRCODE\$ field may be assigned a value.

## \$ Dollar sign

System fields are distinguished from "ordinary" fields by the presence of a dollar sign ("\$\$") as the last character of the field name.

Field	Picture	Definition
AUTHOR\$\$	X(8)	user id of programmer who wrote the TQL program.
CC\$\$	X(2)	Current century.
CCYY\$\$	X(4)	Current year in long format.
CJUL\$\$	9(7)	Current date in CCYYDDD (Julian or day-of-year) format.
CYMD\$\$	9(8)	Current date in CCYYMMDD format.
DAY\$\$	X(20)	Current day of the week as a text field (i.e. Friday)
DD\$\$	X(2)	Current day.
DESC\$\$	X(30)	Description of program from PROGRAM-ID clause.
DMCY\$\$	9(8)	Current date in DDMMCCYY format.
DMY\$\$	9(6)	Current date in DDMMYY format.
ERRCODE\$\$	X(1)	A status field (this field may be set by the program).
GRPCNT\$\$	9(2)	Number of actual groups as returned by TIPGRPS call.
GRP\$\$	X(8) OCCURS 16 TIMES	Table of user's group membership as returned by TIPGRPS call.
HH\$\$	X(2)	Current hour.
HHMM\$\$	9(4)	Current time of day in HHMM format.
JUL\$\$	9(5)	Current date in YYDDD (Julian or day-of-year) format.
KEY\$\$	X(1)	Internal representation of the key pressed. Use the following condition names instead of directly examining this field.  TQL-CANCEL-KEY TQL-CONFIRM-KEY

		TQL-DELETE-KEY
		TQL-MORE-KEY
		TQL-NEXT-KEY
		TQL-PREVIOUS-KEY
		TQL-REFRESH-KEY
		TQL-TIMEOUT-KEY
		TQL-TRANSMIT-KEY
		TQL-UPDATE-KEY
LEVEL\$	9(4)	Binary numeric field specifying the current control level break. It may be used to index into accumulator arrays for control break processing.
LINE\$	9(3)	Current line number (in a report).
LPP\$	9(4)	Binary numeric field containing the number of printable lines on a page. It may be used in report writing.
MIN\$	X(2)	Current minute.
MONTH\$	X(20)	Current Month as a text name (i.e. January)
MON\$	X(2)	Current month.
PAGE\$	9(5)	Current page number (of report).
PROGRAM\$	X(8)	TQL program-id name
SITE\$	X(12)	TIP site name.
TID\$	X(4)	Terminal name where the TQL program is executing.
TIME\$	9(6)	Time of day in HHMMSS format.
UID\$	X(8)	user id of user running TQL program.
YMD\$	9(6)	Current date in YYMMDD format.
YY\$	X(2)	Current year.

---

## TQL Program Structure

The general structure of a TQL program follows this layout:

### **IDENTIFICATION DIVISION.**

```

PROGRAM-ID. name.
DATA DIVISION.
FILE file-name-1.
RECORD record-name-1. *>THIS IS THE RECORD NAME
[ ALLOW CHANGE, NO CHANGE, VERIFY, ID, clauses ]
[ ALLOW DELETE. ] [ NO DELETE. ]
[ ALLOW ADD. ] [ NO ADD. ]
RECORD record-name-n. ...etc...
FILE file-name-n.
RECORD record-name-n. ...etc....
[ WORKING-STORAGE SECTION. ]
[ DECLARATIVES SECTION. ]
*
* comments may be entered anywhere
* by entering an asterisk (*) in column 7
* comments continue to the end of the line
*
[ DISPLAY DIVISION. ]
[ REPORT DIVISION. ]

```

A program may specify as many files and records as are needed for the application. TQL programs may have any number of defined displays and/or reports. There must be at least one display or report in a TQL program.

## IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION of a TQL program must appear first and is required in all TQL programs. This division:

names the TQL program,

may provide an informative description of the program and

may restrict run-time access of the program to specific TIP users.

### Syntax:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. progname
[ "Comments..." ]
[ PRIORITY = n ]
[ "description of program" ]
[ GROUP = id ]
[ GROUPS = ( , , , ) ]
[ MAXREAD n ]
[ PASSWORD PROTECT ]
[ RESET READ FROM ]
[ CURRENCY [ SIGN ] [ IS ] <literal> ]

```

[ DECIMAL-POINT [ IS ] COMMA ]

**Where:**

**programe**

Up to eight characters (the first of which must be alphabetic) which uniquely identifies the program. This name is used by the run-time TQL user to run this TQL program.

**PRIORITY=**

A keyword on the PROGRAM-ID statement that sets the execution priority (number) for that TQL program in the on-line system. This keyword is accepted but has no effect.

**description**

Up to thirty characters (enclosed in single quotes) which provide a description of the program.

This is the character string returned as the system field "DESC\$" (see also "SYSTEM FIELDS").

**GROUP =**

A group name that specifies which users may execute this TQL program. A user may use this program only if their TIP user id or a group to which they belong or their terminal name matches this id.

**GROUPS =**

A list of up to 8 group names that are used to determine which users may execute this TQL program.

A user may use this program only if their TIP user id or a group to which they belong or their terminal name matches an id in this list.

This clause may be specified in addition to or instead of the GROUP= clause.

If neither GROUP= nor GROUPS= is specified, the TQL program is not restricted to any specific users.

**PASSWORD PROTECT**

If this clause is specified, the programmer is asked when the TQL program is compiled) to assign a password for this program.

Whatever password is assigned by the programmer must be supplied by all users who attempt to run this TQL program.

To change the password of a TQL program, the programmer must recompile the program and specify a new password.

**MAXREAD n or MAXREAD = n**

This clause specifies the number of read operations that this program may perform before TQL calls TIPTIMER to temporarily release resources. This also defines how often TQL will check for user interruption via the CANCEL key.

**RESET READ FROM**

This clause allows for READ FROM to reset its sequential position if the identifier that holds the FROM specification changes value although no new driving read has occurred. The default is to not reset sequential position until a new driving record is present.

**CURRENCY SIGN IS literal**

The specified literal will be used as the currency symbol. The literal must be in quotes and can only be one character long. This clause will override the TQL system default.  
Default: "\$"

**DECIMAL-POINT IS COMMA**

This clause indicates that the decimal point is the "," character and the thousands separator is the "." character.  
Default: Off.

**Example:**

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MYTQL 'Comments...' PRIORITY=2.
```

## DATA DIVISION

The DATA DIVISION of a TQL program is a required division and must immediately follow the IDENTIFICATION DIVISION.

The first section of the data division identifies the files and records that are used by the program. Subsequent (optional) sections define program work fields (WORKING-STORAGE SECTION) and exceptional event processing (DECLARATIVES SECTION).

**Syntax:**

```
DATA DIVISION.
FILE file-name.
[ RECORD rec-name. ]
[ ALLOW, VERIFY, ID, clause(s) etc. ]
[ 01 name. ]
```

```
[ 05 ... ]
[ ... ]
[ ALLOW, VERIFY, ID, clause(s) etc. ]
```

**Where:****file-name**

The name of a pre-compiled file description.

There must be at least one file specified in a TQL program.

**rec-name**

The name of a pre-compiled record description.

**name** The record name of an explicitly defined record that is coded inline

**Additional Considerations:**

Records may be defined either by referring to the name of a pre-compiled record (that is, via the RECORD clause), or by actually coding the record description instead of the RECORD clause.

More than one record may be specified for a file; more than one file may be specified in a TQL program.

The ALLOW, VERIFY, and ID clauses may be specified in a pre-compiled record description, after the RECORD clause, or after the in-line record description.

Example of a Data Division Definition

```
DATA DIVISION.
FILE PAYMAST.
RECORD PAY-HDR.
RECORD PAY-DETL.
FILE PAYTRANS.
01 PAYTRAN.
05 FILLER PIC X(4) .
05 PAYTRAN-ID PIC X(2) .
05 PAYTRAN-DATA OCCURS 12 TIMES.
10 PAYTRAN-AMOUNT PIC S9(7)V9(2) .
ALLOW CHANGE ALL
ALLOW DELETE ALLOW ADD
VERIFY PAYTRAN-ID = 'B3'
MUST ADD PAYTRAN-AMOUNT.
```

**WORKING STORAGE SECTION**

The WORKING-STORAGE SECTION of the DATA DIVISION of a TQL program is an optional section that may be included by the programmer to

define work fields that are used in computations or other data manipulations.

The section must contain only a single 01 level. All fields must be subordinate to this group item.

Since VALUE clauses are ignored by TQL, these fields are initialized by TQL to zero or spaces (as appropriate) every time the run-time user issues a run-time command (other than NEXT, PREV or MORE).

The WORKING-STORAGE section may be followed by the attribute lists just like any other record description.

Fields that are defined in the WORKING-STORAGE section may be displayed or reported as if they were fields in a record.

**Example:**

```

WORKING-STORAGE SECTION.
01  WORK-FIELDS .
    05  GRAND-TOTAL          PIC S9(7)V99
                                COMP-3.
    05  SUB-TOTAL            PIC S9(7)V99
                                COMP-3.
    05  FULL-ADDRESS .
        10  FULL-ADDRESS-1  PIC X(40) .
        10  FULL-ADDRESS-2  PIC X(40) .
        10  FULL-ADDRESS-3  PIC X(20) .

```

## TQL Statements

In the following sections, reference is made to the syntax of various TQL statements that may appear in a TQL program.

TQL does not depend on the presence of periods to delimit statements (in this respect TQL is quite unlike standard COBOL-74). In fact, TQL does **not** allow a period except to end:

a SECTION or

a DIVISION or

an "ON" clause in the DECLARATIVES SECTION.

TQL allows the programmer to group more than one statement into statement blocks (for use in loops or IF statements etc.). To group statements into statement blocks (or lists), the programmer can enclose group statements by { ... } pairs:

**Example:**

```

IF (A <> B) {
MOVE 0                TO AMT-OWING

```



```

MOVE 0          TO AMT-OWING-30
MOVE 0          TO AMT-OWING-60
MOVE 0          TO AMT-OWING-90
MOVE 0          TO AMT-OWING-120
}

```

In the above example, the five MOVE statements are grouped so that they are considered a single compound statement that is to be executed if A is not equal to B.

**Example:**

```

DO 10 { READ PART-MASTER-REC
PUT PART-NUMBER
PART-DESCRIPTION
PART-STANDARD-COST
}

```

### PERFORMED procedures

TQL allows the programmer to define procedures (groups of statements) that may be PERFORMed from other areas of the same DIVISION. There are a number of basic rules:

the TQL compiler is a single pass compiler; therefore, a routine cannot be performed unless it has already been declared.

the only statements that are allowed in a routine are those statements that are allowable in that DIVISION.

routine names are unlimited in length but only the first 30 characters are significant.

The syntax is illustrated by the examples which follow.

**Example:**

```

DECLARATIVES SECTION.
ON PERFORM OF INIT-TABLE
IF STATUS-TBL (1) = ' '
{ MOVE 'Active' TO STATUS-TBL (1)
MOVE 'Deleted' TO STATUS-TBL (2)
MOVE 'Temporary' TO STATUS-TBL (3)
MOVE 'Closed' TO STATUS-TBL (4)
}.
ON READ OF CUSTOMER-REC
PERFORM INIT-TABLE
MOVE CUST-STAT TO CS-INDEX
MOVE STATUS-TBL (CS-INDEX) TO W-DESC.

```

In this example, a working storage table is initialized to contain the long descriptions of various record status codes. The status code byte in the record is then moved to a work field (that is defined appropriately as a

binary halfword). The halfword is then used as an index into the table of descriptions.

**Example:**

```

DISPLAY DIVISION.
ON PERFORM OF HDR-INFO
CUST-NUM CUST-NAME CUST-PHONE.
AR: READ CUSTREC
PERFORM HDR-INFO
PUT CUST-BALANCE CUST-OVER-30 CUST-OVER-60
USING SCREEN1.
ADDR: READ CUSTREC
PERFORM HDR-INFO
PUT CUST-ADDR1 CUST-ADDR2 CUST-PROV CUST-
ZIP
USING SCREEN2.

```

This example defines a routine named HDR-INFO that simply displays three fields. The routine is used in both the accounts receivable display (AR) and in the address display (ADDR).

**Example:**

```

REPORT DIVISION.
ON PERFORM OF HEADINGS
HOME$
TAB$(30) 'Inglenet Business Solutions'
TAB$(60) 'Page: ' PAGE$ NL$
80 NL$ NL$.
ON PERFORM OF FOOTINGS
NL$ 80 NL$.
RPT1: PERFORM HEADINGS
...
PERFORM FOOTINGS
ON AUX1.
RPT2: PERFORM HEADINGS
...
PERFORM FOOTINGS
ON PRNTR.

```

In this example a heading procedure and a footing procedure are defined. These procedures can be PERFORMed by any of the various reports that are defined in the REPORT DIVISION. This technique is also useful in situations where headings may need to be output in more than one place in a report.

## DECLARATIVES SECTION

The DECLARATIVES SECTION of the DATA DIVISION of a TQL program is an optional section that may be included by the programmer to define special processing that is to occur after a specified record is read or immediately before a specified record is written or added. For example, the programmer may wish to time stamp all records which are written to a file. Rather than have the user enter the current date and time for each record (a tedious and error-prone procedure), the program could accomplish this by coding the appropriate move statements in the Declaratives section.

### Syntax:

```
DECLARATIVES SECTION.  
[ ON ADD OF record-name statements. ]  
[ ON DELETE OF record-name statements. ]  
[ ON READ OF record-name statements. ]  
[ ON WRITE OF record-name statements. ]  
[ ON PERFORM OF procedure statements. ]
```

### Where:

#### ON READ OF

Clause indicating that the statements which follow are to be executed immediately AFTER any read of the specified record name.

An ON READ declarative for one record may contain READ statements to get other records. This is the only way to select records (by using an IF clause at run time) based on the value of data in supplementary records.

#### ON WRITE OF

Clause indicating that the statements which follow are to be executed immediately BEFORE any write of the specified record name.

#### ON ADD OF

Clause indicating that the statements which follow are to be executed immediately before any new record is added to the file.

If both ON WRITE and ON ADD declaratives exist for the same record, TQL will only execute the ON ADD declarative for added records. The ON WRITE declarative will then only be used when a record is updated (rewritten).

#### ON DELETE OF

Clause indicating that the statements which follow are to be executed immediately before any record is about to be deleted from the file.

#### ON PERFORM OF

Clause indicating defined procedures (groups of

statements) that may be PERFORMed from other areas of the same DIVISION. See PERFORMED procedures for a number of basic rules to be followed.

**record-name**

Specifies the record name associated with this clause.

**statements**

One or more statements which are to be executed.

Valid statements for use in the DECLARATIVES SECTION are described as follows:

```

expr
{ statement-list }
[ DO ] number { statement-list }
ADD expr TO field ...
ADD expr ... TO identifier [ROUNDED] ...
ADD expr ... TO expr GIVING identifier
[ROUNDED]...
BREAK
COMMIT
COMPUTE field = expr
DELETE record
DISPLAY 'string'
DIVIDE expr INTO identifier [ROUNDED] ...
DIVIDE expr INTO expr GIVING identifier
[ROUNDED] ...
DIVIDE expr BY expr GIVING identifier
[ROUNDED] ...
DIVIDE expr BY|INTO expr GIVING identifier
[ROUNDED]
REMAINDER identifier
ERROR 'string' [ AT number ]
EXIT
GET record FROM field
GET record FROM field BY key-field
GET record VIA field
GET record VIA field BY key-field
IF expr { statement-list }
IF expr { statement-list } ELSE {
statement-list }
INSERT record [ AT number ]
MOVE expr TO field
MOVE expr TO identifier
MULTIPLY expr BY identifier [ROUNDED] ...

```

```

MULTIPLY expr BY expr GIVING identifier
[ROUNDED] ...
NEXT-LOOP
NEXT RECORD
ON ERROR 'string'
PUT expr ...
READ record
READ record FOR UPDATE
READ record FROM field
READ record FROM field BY key-field
READ record VIA field
READ record VIA field BY key-field
SET condition-name TO TRUE|FALSE
SUBTRACT expr FROM field ...
SUBTRACT expr ... FROM identifier [ROUNDED]
...
SUBTRACT expr ... FROM expr GIVING
identifier [ROUNDED] ...
SUM expr INTO accumulator-array
UPDATE record
WHILE expr { statement-list }

```

**Where:****expr**

A standard TQL expression.

**DO number {statement-list}**

Indicates that the instructions appearing inside the parentheses are to be repeated the specified number of times. This is the simplest way to display more than one record. The loop will be exited early if any imbedded READ statement fails to retrieve a record from the file.

**ADD** The arithmetic expression is evaluated, added to the value of the field and the result stored in the field.

**BREAK**

Exit a loop early.  
Only valid in a loop construct.

**COMMIT**

All pending record updates are done immediately and TIP will commit all record updates and unlock the record keys. If the system stopped after this point all updates are completed and would not be rolled back.

**COMPUTE**

The arithmetic expression is evaluated, and the result stored in the field.

**DELETE record**

The named record is immediately deleted from its file.

**DISPLAY 'string'**

Used to redisplay the whole screen, possibly with new data.

**DIVIDE expr**

Used for division providing specified remainders.

**ERROR 'string' [ AT number ]**

The specified string is used as an error message and the system field ERRCODE\$ is set to a non-blank value to indicate an error has occurred. This statement will exit the declarative immediately.

This clause may be used to signal an error condition, for example:

```
IF (COUNT LT 10) (ERROR 'NOT ENOUGH STOCK')
```

The optional AT number clause can be used to highlight the field number in error. The FCC attribute ".B" is used.

**EXIT**

The EXIT verb may be used in this section to cause an immediate EXIT from a particular ON condition. For example:

```
ON READ OF CUSTREC
...
IF (AREA-CODE = 416) {EXIT}
...
```

In this example, the statements that precede the area code test are executed for all customer records that are read.

The statements following the area code test, however, are only executed if the area code is not 416. If the area code is 416, the EXIT verb immediately (and prematurely) terminates the declarative (it does not skip or otherwise affect the record that is being processed the NEXT RECORD statement is used to skip the current record).

The main difference between EXIT and NEXT RECORD is whether or not the statements following are executed. In the case of NEXT RECORD, the statements that follow are executed by TQL. In the case of the EXIT verb, the statements that follow are not executed.

**GET record**

Directs TQL to read the specified record for information purposes only. (A READ may actually result in a read for update if an UPDATE, ADD, or DELETE command is being processed, whereas "GET" is always a read for information only).

**IF** The relational expression immediately following the IF is evaluated. If it is found to be true then the code inside the parentheses will be executed. If it is found to be false then the code in parentheses following the ELSE will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause.

**INSERT record [ AT number ]**

Directs TQL to add the specified record to the file immediately. The user program is responsible for ensuring that all fields of the record contain valid data.

If the AT clause is specified the record is added at the specified record number position. This clause is only applicable for direct access (DAM) files. If a record already exists at that location it will be overwritten.

**MOVE**

The expression is evaluated, and the result is stored in the field. TQL now accepts:

MOVE ZERO TO xxx

**MULTIPLY**

Used for expression multiplication.

**NEXT-LOOP**

Go to the test condition of the loop.

**Note:** Only valid in loop construct.

**NEXT RECORD**

This moves an "S" to the field ERRCODE\$, indicating the current record is to be bypassed.

**ON ERROR**

This clause may be used following a read statement to specify a string which will be displayed if an error occurred during the read. If ERRCODE\$ is not a space then the declarative will be exited with the specified literal being used for the error message.

**PUT expr**

Used to signify that the expression is to be sent for output.

The following code is identical:

```
READ MY-REC
DSPLY-1 DSPLY-2 DSPLY-3
```

and

```
READ MY-REC
PUT DSPLY-1 DSPLY-2 DSPLY-3
```

The PUT statement provides consistency in the statements - all statements now start with a verb.

PUT also allows for easy handling of field-names after verbs that take multiple receiving items (MOVE, ADD...)  
For example, instead of:

```
MOVE "A" TO ITEM-1  
{ item-list }
```

you would use

```
MOVE "A" TO ITEM-1  
PUT item-list
```

**Note:** The use of PUT is optional. We recommend that you use PUT since future releases of TQL will enhance the use of the PUT verb. Since the use of PUT removes some ambiguities in the language, its use may become mandatory. Inqlenet supplies a conversion program ([cvttql](#)) for your TQL programs.

#### **READ record**

Directs TQL to read the specified record.

#### **READ record FOR UPDATE**

This command reads the record and imposes a record lock (FCS-GETUP) even if the end user is only processing an inquiry type of command. This allows TQL programmers to update records and is usually followed by an UPDATE or DELETE command.

#### **READ record VIA field**

"field" is the name of a field which contains the key of the record to be read. If the READ is being done because a record is about to be ADDED or UPDATED then the READ is actually a read for update ('FCS-GETUP') and the record will be updated back to the file.

#### **READ record FROM field**

"field" is the name of a field holding (part of) the key for the secondary record. The file is read sequentially until this first portion of the key no longer matches the value in "field".

#### **READ record BY field**

"field" is the name of a field that defines a secondary key for the file being read. This clause may be used to indicate that the file is to be read via a secondary key. The reserved names "KEY1", "KEY2" ... "KEY10" may be used to imply the key location (when it is undesirable or not practical to use a defined field name).

#### **SET condition-name TO TRUE | FALSE**

This verb provides for level 88 support. A condition name can also be set to FALSE if a when false literal is specified.

For example:



```

05 FOO                PICTURE X(4) .
05 FOO-TOO           PICTURE X(4) .
   88 FOO-88-1       VALUE "ABCD" "DEFG" .
   88 FOO-88-2       VALUE "TEST" .
   88 FOO-88-3       VALUE "ABCD"
WHEN FALSE "DCBA" .

```

```

MOVE FOO-88-2        TO FOO ①
MOVE FOO-88-1        TO FOO ②
MOVE FOO-88-3        TO FOO ③
SET FOO-88-2         TO TRUE ④
SET FOO-88-1         TO TRUE ⑤
SET FOO-88-3         TO FALSE ⑥
SET FOO-88-2         TO FALSE ⑦

```

- ① Moves "TEST" into FOO.
- ② Not valid, multiple values in condition name.
- ③ Not valid, multiple values.
- ④ Moves "TEST" into FOO-TOO.
- ⑤ Invalid multiple values.
- ⑥ Moves "DCBA" into FOO-TOO.
- ⑦ Invalid no FALSE literal.

**SUBTRACT**

The arithmetic expression is evaluated, subtracted from the value of the "field" and the result is stored in the field.

**UPDATE record**

The named record is immediately written back to the data file.

**WHILE**

The relational expression immediately following the WHILE is evaluated. If it evaluates to be true, the code inside the parentheses will be executed. The code is executed repeatedly until the expression evaluates to be false.

**Example:**

```

DECLARATIVES SECTION .
ON READ OF PAYMAST ADD PAYMAST-SALARY TO
WS-TOTAL-SALARY
  ADD 1 TO WS-PAYMAST-COUNT .
ON WRITE OF PAYMAST MOVE TIME$ TO PAYMAST-
TIME-WRITTEN
  MOVE YMD$ TO PAYMAST-DATE-WRITTEN .

```

In this example, every time a record named 'PAYMAST' is read, TQL automatically executes the two ADD statements (which presumably modify some WORKING-STORAGE fields for later use).

Immediately before all writes of records named 'PAYMAST', TQL automatically executes the two MOVE statements (which take advantage of the system fields to move the current date and time to corresponding fields in the PAYMAST record).

## DISPLAY DIVISION

The DISPLAY DIVISION of a TQL program is a division that defines the display sets that are available at execution time.

Each display set contains statements that specify the fields that are to be displayed. In addition, the display set contains VERBS that specify (to TQL) exactly which records to read or other operations to be performed.

At execution time, the user of the TQL program uses the name of a display set to request the display of data according to the specifications of the display set.

### Syntax:

```

name : display-list USING mcsname ['fillchar']
[BEFORE [DISPLAY] PROCEDURE [IS] beforeproc]
[AFTER [DISPLAY] PROCEDURE [IS] afterproc]
[FCC DATA [AREA] [IS] fccdata]
[CURSOR DATA [AREA] [IS] cursordata]
[ON ENTER on-enter-name [on-enter-control]]

```

### Where:

**name** The display name. This name is required and must be unique within a TQL program. This name is used at execution time by the TQL user to request a particular display format.

#### **on-enter-name**

is one of the following:

```

display-name
data-name VIA on-enter-proc

```

#### **on-enter-control**

is one of the following:

```

[RETURN [CONTROL]] BEFORE [ADD]
AFTER [ADD]

```

#### **display-list**

is one or more of the following:

```

expr
{ display-list }
[ DO ] number { display-list }
ADD expr TO field
ADD expr ... TO identifier [ROUNDED]
ADD expr ... TO expr GIVING identifier [ROUNDED]

```

```

BREAK
COMPUTE field = expr
DIVIDE expr INTO identifier [ROUNDED] ...
DIVIDE expr INTO expr GIVING identifier [ROUNDED] ...
DIVIDE expr BY expr GIVING identifier [ROUNDED] ...
DIVIDE expr BY|INTO expr GIVING identifier [ROUNDED]
REMAINDER identifier
IF expr { display-list }
IF expr { display-list } ELSE { display-list }
MORE$
MOVE expr TO field
MOVE expr TO identifier
MULTIPLY expr BY identifier [ROUNDED] ...
MULTIPLY expr BY expr GIVING identifier [ROUNDED] ...
NEXT-LOOP
NL$
PUT expr ...
READ record
READ record BY key-field
READ record FROM field
READ record FROM field BY key-field
READ record VIA field
READ record VIA field BY key-field
SET condition-name TO TRUE|FALSE
SUBTRACT expr FROM field
SUBTRACT expr ... FROM identifier [ROUNDED]
SUBTRACT expr ... FROM expr GIVING identifier [ROUNDED]
WHILE expr
    
```

### USING mcsname

This is the name of the (initial) TIP screen format that is to be used to control the display format of the data. The screen format name may be coded with or without quotes.

### fillchar

You can specify a "\*" or an "\_" as the optional fill character for entry fields in the specified screen format (mcsname).

### BEFORE DISPLAY PROCEDURE IS beforeproc

The procedure beforeproc is executed before the display code is executed. If ERRCODE\$ is not space when beforeproc returns, the display will not be run. Typical uses for the BEFORE PROCEDURE are to set field attributes for a screen or to limit the number of times a record can be added in enter mode.

### AFTER DISPLAY PROCEDURE IS afterproc

The procedure afterproc is executed when the user presses TRANSMIT, MSG-WAIT, any function key or after a time out has occurred for the associated screen. The procedure gets control before TQL does any processing including automatic VERIFY and NO CHANGE processing. IF ERRCODE\$ is not space when the procedure returns TQL will not continue its processing and return to the screen display in the same mode (that is update, enter etc.)

Typical uses for the AFTER PROCEDURE are to do further validation of user input or to prevent certain actions such as an attempt to update by a given set of users.

The system field KEY\$ and its associated condition names may be used to query which key was pressed. We recommend using the condition names instead of looking at the value of KEY\$. The content of KEY\$ is an internal representation of the pressed key and could be subject to change in future releases. Do not rely on values observed in KEY\$.

**FCC DATA AREA IS fccdata**

The specified data area will be used as the FCC table for the screen.

fccdata must be an alphanumeric data item defined in the WORKING-STORAGE section. TQL does not do any validation of the size of this data item. It is the programmer's responsibility to ensure that it is large enough to account for all the fields on the screen.

**CURSOR DATA IS cursordata**

The specified data area will be used as the cursor modification table for the screen.

cursordata must be an alphanumeric data item defined in the WORKING-STORAGE section. TQL does not do any validation of the size of this data item. It is the programmer's responsibility to ensure that it will be large enough to account for all the fields on the screen.

**ON ENTER displayname****ON ENTER dataname VIA displayproc**

At the end of the DISPLAY definition, you can add the ON ENTER clause. If you used the ENTER command to get to this first display, once you have completed entering the first screen full of data and press TRANSMIT, TQL will switch to the displayname specified in the ON ENTER clause. This syntax is useful for entering a header record followed by several detail records. See the "ORD" Program Description example in this manual. To end this function, press CANCEL.

If the second form of the ON ENTER clause is used the display that is run can be set at run-time. The procedure displayproc will be executed before TQL starts the ENTER screen. The display name to be run should be moved into dataname. When the procedure exits, TQL will run the display in this field. If the field is spaces then no enter processing will occur.

ON ENTER processing can be chained together. A display run as part of ON ENTER processing may itself perform ON ENTER processing.

```
[ RETURN CONTROL ] BEFORE ADD  
AFTER ADD
```

These two clauses determine when the parent record is actually written. The record in the display that has the ON ENTER clause is considered the parent record. Records for displays run as part of the ON ENTER processing are the child records.

The second clause is the default behavior. The ENTER processing occurs after the record for the display invoking the ON ENTER processing has been written.

The BEFORE ADD clause causes the ON ENTER processing to occur for the children before the parent is written. When the ON ENTER processing has completed the parent record is automatically written. There is one parent written for every invocation of the ON ENTER processing.

The RETURN CONTROL is only applicable to the BEFORE ADD clause. This clause prevents TQL from automatically writing the parent record until the parent screen is exited using MsgWait. This allows for running multiple ON ENTER displays followed by one summary parent record.

**field** The name of a data field. If the field is part of an OCCURS clause, it may be followed by the occurrence number, for example: PART-NUM(3). If no occurrence number is given, the first occurrence is assumed. The field name must be subscripted by a numeric expression. A subscript field may be part of a record structure or a working-storage field

**[DO] number {display-list}**

Indicates that the instructions appearing inside the parentheses are to be repeated the specified number of times. This is the simplest way to display more than one record. The "loop" will be exited early if any READ statement fails to retrieve a record from the file.

**expr** A standard TQL expression (see TQL Expressions).

**ADD** The arithmetic expression is evaluated, added to the value of the "field" and the result stored in the "field".

Note: The result fields as well as fields in the expression may be subscripted when appropriate by either a constant or subscript field.

**BREAK**

Exit a loop early.

Note: Only valid if in a loop construct.

**COMPUTE**

The arithmetic expression is evaluated, and the result stored in the "field".

**DIVIDE**

Used for division providing specified remainders.

**IF** The relational expression immediately following the IF is evaluated. If the expression is true the code inside the parentheses will be executed. If the expression is false the code in parentheses following the ELSE will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause. It may be necessary to enclose the expression in parentheses to avoid confusion with subscripting.

### **MORE\$**

Marks the point from which the display is to be continued when more detail records are requested at execution time. The TQL user can request "more" records by entering the "MORE" run-time command (or by pressing F9).

### **MOVE**

The expression is evaluated, and the result is stored in the "field".

The MOVE verb allows data to be moved from an alphanumeric field (PIC X) to a strictly numeric field (PIC 9). Digits from the alphanumeric field are moved (using right justification) into the numeric field. No decimal point alignment is performed since decimal point alignment is undefined in an alphanumeric field.

### **MULTIPLY**

Used to multiply.

### **NEXT-LOOP**

Go to the top of a loop. Involves loop counter increment. Note: Only valid in a loop construct.

**NL\$** This notation may be inserted to indicate (to the automatic screen generation process see TQLMON "M" command) that the screen format is to start a new line at this point. This NL\$ specification has no other effect.

### **PUT expr**

Used to signify that the expression is to be sent for output. The following code is identical:

```
READ MY-REC
DSPLY-1 DSPLY-2 DSPLY-3
```

and

```
READ MY-REC
PUT DSPLY-1 DSPLY-2 DSPLY-3
```

The PUT statement provides consistency in the statements all statements now start with a verb.

PUT also allows for easy handling of field-names after verbs that take multiple receiving items (MOVE, ADD...)

For example, instead of:

```
MOVE "A" TO ITEM-1
{ item-list }
```

you would use

```
MOVE "A" TO ITEM-1
PUT item-list
```

Note: The use of PUT is optional. We recommend that you use PUT since future releases of TQL will enhance the use of the PUT verb. Since the use of PUT removes some ambiguities in the language its use may become mandatory. IngleNet supplies a conversion program (cvttql) for your TQL programs.

**READ record BY field**

field is the name of a data item that defines a key for the file being read. This clause is be used to indicate that the file is to be read via an alternate key. The reserved names KEY1 ... KEY10 may be used to imply the key location (when it is undesirable or not practical to use a defined field name).

**READ record VIA field**

field is the name of a data item which contains the key of the record to be read.

**READ record FROM field**

field is the name of a data item holding (part of) the key for the secondary record. The file is read sequentially until this first portion of the key in the record no longer matches the value in field.

**SET condition-name TO TRUE|FALSE**

This verb provides for level 88 support. A condition name can also be set to FALSE if a when...false literal is specified.

For example:

```
05 FOO                PICTURE X(4) .
05 FOO-TOO           PICTURE X(4) .
    88 FOO-88-1       VALUE "ABCD" "DEFG" .
    88 FOO-88-2       VALUE "TEST" .
    88 FOO-88-3       VALUE "ABCD"
WHEN FALSE "DCBA" .
```

```
MOVE FOO-88-2        TO FOO ①
MOVE FOO-88-1        TO FOO ②
MOVE FOO-88-3        TO FOO ③
SET FOO-88-2         TO TRUE ④
SET FOO-88-1         TO TRUE ⑤
SET FOO-88-3         TO FALSE ⑥
SET FOO-88-2         TO FALSE ⑦
```

- ① Moves "TEST" into FOO.
- ② Not valid, multiple values in condition name.
- ③ Not valid, multiple values.
- ④ Moves "TEST" into FOO-TOO.
- ⑤ Invalid multiple values.
- ⑥ Moves "DCBA" into FOO-TOO.
- ⑦ Invalid no FALSE literal.

### **SUBTRACT**

The arithmetic expression is evaluated, subtracted from the value of the field and the result is stored in the field.

### **WHILE**

The relational expression immediately following the WHILE is evaluated. If the expression is true, the code inside the parentheses will be executed. The code is executed repeatedly until the expression evaluates to be false.

#### **Example:**

```

DEPLST: READ DEPT-REC, DEPT-NUM, DEPT-NAME,
MOVE 0 TO TOT-SAL,
MORE$
DO 19 {READ PAYREC FROM DEPT-NUM,
NAME, SIN, SALARY, NL$
ADD SALARY TO TOT-SAL,
}
TOT-SAL
USING PAYSCRN.

```

In the above example, TQL does the following for each display:

read a department record (DEPT-REC) and display the fields DEPT-NUM and DEPT-NAME.

collect up to 19 payroll records (PAYREC) which are in the selected department. The payroll file has the department number as the first part of the key of the record.

for each PAYREC the fields NAME, SIN, and SALARY are displayed.

SALARY is accumulated in the field TOT-SAL.

TOT-SAL is the last field displayed on the screen

The Message Control System (MCS) screen format name is PAYSCRN. Note that the inclusion of the NL\$ notation forces the automatic screen generator to begin a new line in the screen format at that point.

If more than 19 payroll records exist then the terminal operator may ask for more by pressing the MORE key or entering the run-time command MORE. TQL will continue from the point marked by the tag: MORE\$.



**Example:**

```

DEPSUM: DO 20 {
  READ PAY-REC, READ DEPT-REC VIA DEPT-NUM,
  NAME, SIN, SALARY, DEPT-NAME NL$
}
USING PAYDEPT.

```

In the preceding example, TQL does the following for each display:

read a payroll record (PAY-REC)

then read from the department file (DEPT-REC) by using the field DEPT-NUM as a key. DEPT-NUM must be a field in the PAY-REC record.

for each PAY-REC the fields NAME, SIN, SALARY, and DEPT-NAME (from department record) are displayed.

The Message Control System (MCS) screen format name is PAYDEPT.

**Example:**

```

DISPLAY DIVISION.
ON PERFORM OF AFTERPROC
IF TQL-UPDATE-KEY
AND UID$ = "DEFAULT"
ERROR "You are not allowed to update
records"
ELSE
MOVE SPACE TO ERRCODE$.
TESTDS: READ PAY-REC
PUT PAY-KEY
USING PAYDEPT
AFTER DISPLAY PROCEDURE IS AFTERPROC.

```

This example displays use of the AFTER PROCEDURE to prevent a user named "DEFAULT" from updating records. All other users are allowed to update records.

**Note:** The user DEFAULT would not be prevented from deleting a record if the record allowed deletes.

**REPORT DIVISION**

The REPORT DIVISION of a TQL program is a division that is a TQL extension to standard COBOL. This division defines one or more reports that are available at run time to the TQL user. Each report has an assigned name that is used by the user to select the report. The report definition defines the contents of a logical page of the physical report.

A logical page may consist of more than one physical page. The runtime TQL interpreter will generate the report by repeatedly generating the logical page until no more records are available.

The default destination of the report may be either the site printer (example: PRNTR) or an auxiliary printer. The printout is actually routed by TQL via the TIP printing facility (TIPPRINT). The user may override the destination of the report at the time the report is requested.

### Syntax:

```
name : report-list ON printrname [WITH|NO
HEADING]
[AT END report-list].
```

Where report-list is one or more of the following:

```
expr ...
{ report-list }
[DO] number { report-list }
ADD expr TO field ...
ADD expr ... TO identifier [ROUNDED] ...
ADD expr ... TO expr GIVING identifier [ROUNDED] ...
BREAK
COMMIT
COMPUTE field = expr
DELETE record
DIVIDE expr INTO identifier [ROUNDED] ...
DIVIDE expr INTO expr GIVING identifier [ROUNDED] ...
DIVIDE expr BY expr GIVING identifier [ROUNDED] ...
DIVIDE expr BY|INTO expr GIVING identifier [ROUNDED]
REMAINDER identifier
HOME$
IF expr { report-list }
IF expr { report-list } ELSE { report-list }
INSERT record
MOVE expr TO field
MULTIPLY expr BY identifier [ROUNDED] ...
MULTIPLY expr BY expr GIVING identifier [ROUNDED] ...
NEXT-LOOP
NL$
PUT expr ...
READ record
READ record FOR UPDATE
READ record BY key-field
READ record FROM field
READ record FROM field BY key-field
READ record VIA field
READ record VIA field BY key-field
SET condition-name TO TRUE|FALSE
SKIP$(number)
SUBTRACT expr FROM field ...
SUBTRACT expr ... FROM identifier [ROUNDED] ...
SUBTRACT expr ... FROM expr GIVING identifier [ROUNDED] ...
SUM expr INTO accumulator-array
TAB$(number)
UPDATE record
WHILE expr { report-list }
```

### Where:

#### name

The report name. This must be unique within a TQL

program. At execution time the user will request the production of this report by referring to this report name.

**ON printer**

The clause establishes a default report destination. This clause is mandatory.

Any valid printer name (as accepted by the TIPPRINT interface) may be specified, for example:

**AUX0, AUX1, etc.**

**NO HEADING**

This clause suppresses the automatic heading page that TQL generates to identify the originator of the report. This automatic heading contains the date and time, the originating terminal name and the TQL command that produced the report.

**field** The name of a data field. If the field is part of an OCCURS clause it may be followed by the occurrence number such as PART-NUM(3). If no occurrence number is given then the first occurrence is assumed. The field name may also be subscripted by some other field. A field used as a subscript must be a binary halfword (i.e. PIC 9(4) COMP-4). A subscript field may be part of a record structure or working-storage field.

**DO number {report-list}**

Indicates that the instructions coded inside the parentheses are to be repeated the specified number of times. This is the simplest way to process several records. The "loop" will be exited early if any imbedded READ statement fails to retrieve a record from the file.

**expr** A standard TQL expression (see TQL Expressions ).

**ADD** The arithmetic expression is evaluated, added to the value of the "field" and the result is stored in the "field".

**BREAK**

Exit a loop early.

Note: Only valid in a loop construct.

**COMMIT**

All pending record updates are done immediately and TIP will commit all record updates and unlock the record keys. If the system stopped after this point all updates would be completed and would not be rolled back.

**COMPUTE**

The arithmetic expression is evaluated, and the result stored in the field.

**DELETE record**

The named record is immediately deleted from its file.

**DIVIDE**

Used for expression division producing specified remainder.

**HOME\$**

Force a skip to a new page (top of form). The system field "PAGE\$" is incremented by one and the system field "LINE\$" is set to zero.

**IF**

The relational expression immediately following the IF statement is evaluated. If the expression evaluates to true the code which follows will be executed. If the expression evaluates to false the code which follows the word ELSE will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause.

**INSERT record**

Directs TQL to add the specified record to the file immediately. The user program is responsible for ensuring that all fields of the record contain valid data.

**MOVE**

The arithmetic expression is evaluated and the result is stored in the "field".

**MULTIPLY**

Used for expression multiplication producing specified remainder.

**NEXT-LOOP**

Go to the top of loop. Involves loop counter increment. Note: Only valid in a loop construct.

**NL\$**

Force a new line. The current contents of the print line are printed. The system field LINE\$ is incremented by one.

**PUT expr**

Used to signify that the expression is to be sent for output. The following code is identical:

```
READ MY-REC
DSPLY-1 DSPLY-2 DSPLY-3
```

and

```
READ MY-REC
PUT DSPLY-1 DSPLY-2 DSPLY-3
```

The PUT statement provides consistency in the statements all statements now start with a verb.

PUT also allows for easy handling of field-names after verbs that take multiple receiving items (MOVE, ADD...)

For example, instead of:

```
MOVE "A" TO ITEM-1
{ item-list }
```

you would use:

```
MOVE "A" TO ITEM-1
PUT item-list
```

**Note:** The use of PUT is optional. We recommend that you use PUT since future releases of TQL will enhance the use of the PUT verb. Since the use of PUT removes some ambiguities in the language its use may become mandatory. IngleNet supplies a conversion program (cvttql) for your TQL programs.

**READ record**

Read the specified record name.

**READ record FOR UPDATE**

This command reads the record and imposes a record lock (FCS-GETUP) even if the end user is only processing an inquiry type of command. This allows TQL programmers to update records and is usually followed by an UPDATE or DELETE command.

**READ record BY field**

field is the name of a data item that defines a key for the file being read. This clause may be used to indicate that the file is to be read via an alternate key. The reserved names KEY1 ... KEY10 may be used to imply the key location (when it is undesirable or not practical to use a defined field name).

**READ record VIA field**

field is the name of the data item containing the key of the desired record.

**READ record FROM field**

field is the name of the data name containing (part of) the key for the secondary record. The file is read sequentially until this first portion of the key in the record no longer matches the value in field.

**SET condition-name TO TRUE|FALSE**

This verb provides for level 88 support. A condition name can also be set to FALSE if a WHEN...FALSE literal is specified.

**SKIP\$(number)**

TQL advances the output position (horizontally) to the right by the number of columns indicated. Number must be greater than or equal to 0 and less than or equal to 255.

**SUBTRACT**

The arithmetic expression is evaluated, subtracted from the value of the field and the result stored in the field.

**SUM expression INTO accumulator-array**

Accumulate totals in an accumulator array (defined in WORKING STORAGE).

Used for accumulating totals of level breaks or control breaks.

Any accumulator arrays specified in SUM statements are cleared as required automatically by execution of CONTROL FOOTING code.

**TAB\$(number)**

TQL positions the output pointer into the print line to the exact column specified by number. This statement may position the output pointer after the current column location OR before the current column location. The programmer is responsible for the results of overlapped fields. Number must be greater than or equal to 0 and less than or equal to 255.

**UPDATE record**

The named record is immediately written back to the data file.

**WHILE**

The statements in parentheses following the WHILE are executed repeatedly until the relational expression is false.

**AT END**

When all records have been processed the coding following the AT END clause will be executed. This clause is optional. This provides the capability to generate final totals or summary information etc. This clause must appear as the last clause in a report definition. If the TQL command line begins with AT END the command is deferred until the end user does a CLOSE command and a selection file is active. The end user may then do a record selection using the SELECT and/or SORT commands and then CLOSE TQL. The command as entered from the command line would look like the following:

```
TIP?>OPEN TSTTQL AT END EXPORT (COMPANY MGR TELEPHONE) ON
PRNTR
```

**Note:** This function should only be invoked under some application program control.

**Example:**

```
REPORT DIVISION.
QOH: HOME$
TAB$(15) 'PART - QUANTITY ON HAND'
TAB$(70) 'PAGE' PAGE$ NL$
'PART NUMBER'
```

```
TAB$(20) 'DESCRIPTION'  
TAB$(50) 'QUANTITY' NL$  
DO 50 {  
  READ PARTFIL,  
  PM-NUM  
  TAB$(20) PM-DESC  
  TAB$(50) PM-QTY NL$  
}  
ON AUX1.
```

For each logical page of this report the following is done:

- a new page is forced (HOME\$)
- a two line page title is printed.
- up to 50 PARTFIL records are read.
- for each record the fields PM-NUM, PM-DESC and PM-QTY are printed on a separate line.

The default destination of the report is AUX1. The TQL user may override this at execution time.

If NL\$ was omitted, all 50 print lines would be constructed in the print line buffer and one line representing the destructive intersection of all the data would be printed instead of 50 lines!.

### EDIT\$ - Report Field Editing

The EDIT\$ function provides a broad range of editing capabilities.

#### Syntax:

```
EDIT$(field,mask[,startcol])
```

#### Where:

##### field

The name of the field that is to be edited. This field may be a numeric or alphanumeric field. Subscripted fields are also permitted.

##### mask

A character literal (enclosed in quotes) that defines the edit mask that is to be used to edit the field. More information about the possible edit masks follows this syntax description.

##### startcol

An optional numeric literal that specifies the desired starting column of the edited field in the current report line. If this parameter is omitted, the result of the EDIT\$ operation will begin in the current output column. If this parameter is specified, the EDIT\$ output will begin in the column specified.

For numeric fields, the possible editing specifications are similar to those provided by standard COBOL and the TIP Message Control System (MCS).

Numeric fields are decimal place aligned with respect to the edit mask. Extraneous digits to the left of the decimal place are truncated on the left; decimal digits are rounded to the specified number of decimal places.

The following table shows various example masks along with example input and output results:

Mask	Input	Output	Description
"ZZZ,ZZ9.99"	1234.567	" 1,234.57"	Zero suppression. Comma and decimal place insertion. Decimal place rounding
"-ZZ,ZZ9"	-45	" -45"	Leading minus sign
"ZZ,ZZ9-"	-123	" 123-"	Trailing minus sign
"(ZZZ,ZZ9.99)"	-145.3	" (145.30)"	Floating parentheses for negative values
"(\$ZZ,ZZ9.99)"	-145.3	" (\$145.30)"	Floating parentheses and \$ for negative values
"\$**, **9.99"	1234.56	"\$*1,234.56"	Check protection with \$
"***, **9.99"	1234.56	"**1,234.56"	Check protection without \$
"ZZZ,ZZ9.99CR"	-1234.56	" 1,234.56CR"	CR suffix for negative values
"ZZZ,ZZ9.99DB"	-1234.56	" 1,234.56DB"	DB suffix for negative values
"Total: Z,ZZ9.99"	876.54	"Total: 876.54"	Leading commentary
"99/99/99"	871115	"87/11/15"	Date format
"99:99:99"	120800	"12:08:00"	Time format
"X X X"	"ARC"	"A R C"	Alphanumeric character selection

The basic rules for EDIT\$ are summarized as follows:



A mask character of "Z" or "9" represents a digit selector for numeric fields; the former implies zero suppression, the latter displays the digit regardless of the value.

A leading minus sign - will "float" to the left of the first significant digit or fill character if the numeric field being edited is negative.

A trailing minus sign in the mask will be output to the right of the right most digit if the numeric field being edited is negative; otherwise a space will appear.

If the edit mask is surrounded by parentheses and the edited value is negative, the left parenthesis will "float" to the left of the first significant digit or fill character and the right parenthesis will be output. If the edited field is positive the right parenthesis will be replaced by a space.

A trailing "CR" or "DB" in the mask will be output to the right of the right most digit if the numeric field being edited is negative; otherwise, two spaces will appear.

A mask character of "X" represents a character selector (for alphanumeric fields).

Extraneous characters in the edit mask that follow the last digit or character selector are ignored.

Edit masks are either alphanumeric or numeric. If the mask contains an "X" it is alphanumeric otherwise it is taken as numeric. If you wish to use any special editing characters as normal text use the backslash (\) character to escape the special meaning. (i.e. "TEXT: XXXX" will only substitute for the last four X's.)

For check protection with or without \$, "Z" and "\*" are mutually exclusive according to the COBOL-85 standard.

## Report Heading and Footings

### Report Heading

This defines what is printed or displayed as the first part of the report. It can be any valid TQL function and may, for example:

- print the first page of a report,
- initialize some data fields,
- read an initial set of records or,
- perform another start up procedure.

#### Example:

```
REPORT HEADING {  
DO 25 {"***"} NL$  
'Start of Customer Name and Address Listing' NL$  
' on ' EDIT$(YMMDD$, '99/99/99') NL$
```

```
DO 25 {"**"} NL$
}
```

## Report Footing

This defines what is printed or displayed as the last part of the report. This can be any valid TQL function. You might use this function to display some information accumulated during the report generation.

### Example:

```
REPORT FOOTING {
DO 25 {"**"} NL$
'End of Customer Name and Address Listing' NL$
TTLNUM ' customers printed' NL$
DO 25 {"**"} NL$
}
```

## Page Heading

This defines what is printed or displayed at the top of every page of the report. The page heading procedure is performed on page overflow or after any HOME\$ command. The page heading logic will work regardless of the number of lines on a page. For example, a batch report may have 66 lines per page while printing to AUX0 (the interactive terminal) will only have 24 lines on a page.

### Example:

```
PAGE HEADING {
TAB$(10) 'Page ' PAGE$
TAB$(30) 'Name and Address Listing' NL$
'Number' TAB$(15) 'Name'
TAB$(45) 'Address' NL$ NL$
}
```

## Page Footing

This defines what is printed or displayed at the bottom of every page of the report. This procedure is performed just before the bottom of the page is reached or just before any HOME\$ command.

### Example:

```
PAGE FOOTING {
TAB$(10) 'Bottom of Page ' PAGE$ NL$
}
```

## Control Breaks

### Control Heading

A control heading defines what happens at the beginning of a set of records with a new value in a specified field.

The order in which CONTROL HEADINGS is defined is important. The order defines the major to minor order of the break level.

**Example:**

```
CONTROL HEADING ON state {
  'Listing for the state ' STATE NL$ *> LEVEL$ will be 2
}
CONTROL HEADING ON city {
  ' listing for city ' CITY NL$ *> LEVEL$ will be 1
}
```

The above example illustrates records that are ordered by STATE, and by CITY within STATE.

### Control Footing

A control footing defines what happens at the end of a set of records with a new value in a specified field.

The order in which CONTROL FOOTINGS are defined is important. The order defines the minor to major order of the break level.

**Example:**

```
CONTROL FOOTING ON state {
  'Total sales for state ' STATE ' is ' TTL-SALES (LEVEL$)
  NL$
}
CONTROL FOOTING ON city {
  ' Total sales for ' CITY ' is ; TTL-SALES (LEVEL$) NL$
}
```

The field LEVEL\$ is set to 1 for indexing an accumulator array to obtain necessary totals. LEVEL\$ is set to 2 the level is set according to the sequence of field usage.

At the end of control break processing all accumulator arrays used in the report are reset to zero for every table entry up to and including the value of LEVEL\$.

### Control Footing on Final

A control final defines what happens at the end of the report.

**Example:**

```
CONTROL FOOTING ON FINAL {
  'Grand total all sales is ' TTL-SALES (LEVEL$) NL$
}
```

## TQL Execution Cycle

This section outlines (in point form) the cycle performed by the TQL runtime interpreter for the various commands given to it. This may help programmers understand how to use the Declaratives section effectively.

## READ CYCLE

Step	Description
Step 1:	FCS-SETL to supplied key value, or last key if continuing an inquiry.
Step 2:	If continuing and using secondary index then skip over records until same primary key as last time through.
Step 3:	Get next record.
Step 4:	If end of file or TO key reached, STOP.
Step 5:	If record ID clause is false, go to Step 3.
Step 6:	Perform ON READ coding.
Step 7:	If NEXT RECORD set by ON READ, go to Step 3.
Step 8:	If the IF clause is false, go to Step 3.
Step 9:	Perform any command line MOVEs.

## UPDATE CYCLE

Step	Description
Step 1:	Perform READ (see READ CYCLE above.)
Step 2:	FCS-GETUP record.
Step 3:	Move fields to MCS area, computing expressions (MOVE, ADD, etc.)
Step 4:	Display screen with underscores.
Step 5:	Get reply.
Step 6:	Move fields from MCS area to record checking VERIFY clauses.
Step 7:	If any verify errors then send error message and go to Step 5.
Step 8:	Perform ON WRITE (all READS are with update).
Step 9:	If any errors go to Step 5.
Step 10:	FCS-PUT all records read with update. One record of each type can be held.

---

## TQL Interface to DMS

TQL can access DMS database structures, either by themselves, or in combination with conventional files.

To use TQL in this manner requires TIP/dbi and the schema definition utility [schema](#). The schema must already exist before trying to use TQL to access the database.

This section addresses the various aspects of the TQL/DMS environment and is intended to supplement the "base" TQL documentation.

You should be familiar with:

The TQL environment, and with the concepts of DMS, especially with regard to the online environment.

## SCHEMA Definition

In a non-DMS TQL environment, the normal practice is to precompile a file definition and record definition. To access a DMS database, you must precompile a schema definition, which acts as the TQL file and which also contains all the possible records which can be accessed through that schema.

With conventional files, the programmer can optionally code the record definition in the TQL program. When utilizing the TQL/DMS environment, you must use precompiled records. The record can not be defined in the program. However, the record and schema (file) information is automatically created as a consequence of using the TIP/dbi schema compiler. All that is required by TQL is to define the schema to TQL. This is done using the TQLMON AS command (See [AS - Add Schema](#) ).

## TQL/DMS Programming

Although TQL programs may intermix DMS and conventional file I/O, in this document the aspects of DMS programming are separated from the "base" TQL documentation, for clarity. Only those TQL program Divisions and Sections which are affected are discussed.

Various TQL/DMS statements require the use of a set name. These set names are always coded as character literals where the literal is the actual name of the set being used. For example: 'PART-SET'.

Literals used as set names must be a valid set name from the DMS schema being referenced as a TQL FILE. Literal set names cannot exceed 16 characters in length. Trailing spaces need not be supplied.

TQL will attempt to validate any set names used in a program; however, it is possible to a set name to be valid at compile time but not a run time.. If incorrect set names are used, an appropriate error condition is issued when the executing TQL program attempts to use the incorrect name.

## TQL/DMS: DATA DIVISION

### Syntax:

```

FILE file-name.
[ RECORD rec-name. ]
[ ALLOW, VERIFY, ID, clause(s) ]
[ ALLOW DELETE ]
[ ALLOW DELETE SELECTIVE ]
[ ALLOW DELETE ONLY ]
[ ALLOW DELETE ALL ]

```

As discussed earlier, the "file-name" specified here is the name of the Schema which has been compiled with the TIP/dbi schema compiler. Only those records in the Schema which are to be used in this TQL program need be referenced. All records must be precompiled.

Although conventional file record names are limited to eight characters by TQL, names of DMS records in TQL may be up to 16 characters (the length supported by the DMS pre-compiler). The same [ALLOW](#), [VERIFY](#), etc., clauses that are applicable to conventional records, apply to DMS records as well.

The ALLOW DELETE clause can take one of four different forms. The form instructs TQL which type of DMS DELETE it should issue against the named record, when a DELETE request is given.

The meaning of each DELETE variation is fully described in standard DMS documentation.

## TQL/DMS: DECLARATIVES

DECLARATIVE clauses allow the programmer to designate certain functions that are to be done as a side effect of an **ADD**, **DELETE**, **READ**, or **WRITE** of a particular TQL record. These same clauses are applicable to DMS records as well.

The TQL verbs ADD, READ, and WRITE are the equivalent of the DMS verbs STORE, FETCH, and MODIFY.

The following statements are TQL/DMS statements which may be issued in the DECLARATIVES section, in addition to statements already defined in the "base" TQL documentation.

### Syntax:

```

IF | IFDMS record [NOT] MEMBER 'set-name' {stmtnt-list}
IF | IFDMS record [NOT] MEMBER 'set-name' {stmtnt-list}
[ ELSE {stmtnt-list} ]
IF | IFDMS record [NOT] EMPTY 'set-name' {stmtnt-list}
IF | IFDMS record [NOT] EMPTY 'set-name' {stmtnt-list}
[ ELSE {stmtnt-list} ]
INSERT record INTO 'set-name'
READ record FROM field [BY key-field]
READ record VIA field [BY key-field]

```

```
READ record USING field
READ record OWNER 'set-name'
READ record SET [PRIOR] 'set-name'
READ record AREA
READ record CURRENT
READ record SORTED 'set-name' VIA field
REMOVE record FROM 'set-name'
```

**Note:** "GET" may replace "READ" in the statements above.

**Where:**

**GET record ...**

As with conventional TQL file I/O, use of GET directs TQL to read the specified record for informational purposes only.

Records which are READ will participate in UPDATE operations that occur at runtime, whereas those records retrieved with a GET are strictly retrieved for information only.

**IFDMS record [NOT] MEMBER 'set-name'**

The test of SET membership of record within 'set-name' will be made.

If true, the statement-list inside the parentheses will be executed.

If false, the statement-list in the parentheses following the ELSE will be executed.

If no ELSE clause is given, TQL continues with the next statement after the IFDMS clause.

Nested IFDMS and/or IF clauses are supported to a depth of 10.

**IFDMS record [NOT] EMPTY 'set-name'**

Similar to the MEMBER IFDMS statement.

This statement tests whether the 'set-name' owned by record is empty.

Unlike a regular DML IF EMPTY statement, the set owner record name is required for the TQL IFDMS EMPTY statement.

**INSERT record INTO 'set-name'**

This statement is used to do a DMS INSERT of record into 'set-name'.

The record which owns this set must have been read prior to the time the insert is attempted.

These INSERT statements are only processed during TQL operations in which the named record is either being updated or added to the database.

More importantly, regardless of which DECLARATIVE "ON" clause an INSERT is used in, it will always be deferred until just after the named record has been updated or added to the database.

**READ record FROM field [BY key-field]**

This statement may only be used against DMS records which have been defined to DMS with INDEX keys.

'field' is the name of a field holding (all or part of) the key for a secondary record being processed.

The index structure is read sequentially, and records are returned until this first portion of the key in the record no longer matches the value in 'field'.

The optional BY clause may be used to designate the name of the key field to be used, if other than key number one is desired.

**READ record VIA field [BY key-field]**

This statement may be used on DMS records which either have a CALC key or INDEX keys.

'field' is the name of a field which contains the key of the record to be read.

If the READ is being done because a record is about to be ADDED or UPDATED, then the READ will result in a lock being placed on the record in question, and the record will participate in the update operation when it occurs.

The optional BY clause may be used to designate the key-field to be used in the operation, if other than the default is to be used.

The default for CALC records is the CALC record key, the default for records whose location mode is "indexed", is key 1.

**READ record USING field**

This statement is used to retrieve a record by its actual database key (equivalent to DMS FORMAT 1).

'field' must be a data field defined as S9(8) COMP (The standard database key format).

**READ record OWNER 'set-name'**

This statement instructs TQL to retrieve the named record, which is defined to DMS as the owner of 'set-name'.



The MEMBER record of the set should have just previously been read (with some other READ/GET statement).

If the "just read" record does not currently participate as a MEMBER of the named set, TQL will react as it would with any other "no-find" situation.

#### **READ record SET [PRIOR] 'set-name'**

This statement instructs TQL to retrieve the record which is a MEMBER of 'set-name'.

The default is to retrieve in the NEXT of SET direction; the optional word "PRIOR" can be used to retrieve in the opposite direction.

The set being used must be defined with prior pointers in order to use the PRIOR option.

When using this statement as a "secondary" read on a DISPLAY, set "continuity" is automatically maintained from "screen to screen" by TQL (such as when utilizing a MORE\$ loop on a DISPLAY).

NO special coding is required to reestablish set positioning at the beginning of each MORE\$ loop.

#### **READ record AREA**

This statement instructs TQL to retrieve records sequentially from the DMS AREA in which they reside.

Each execution of this command results in the next logical named record being retrieved, regardless of what other activity may have taken place in this same DMS AREA.

When retrieving records by AREA with DMS, you must initially retrieve the FIRST record of the area, and then request the NEXT record of the area on all subsequent retrievals. This detail is handled automatically by TQL.

#### **READ record CURRENT**

This statement should not be used in the majority of TQL programs, as record currency is re-established, as needed, by TQL.

This is a special purpose statement which is used in rare instances where ambiguity prevents TQL from correctly handling currency considerations automatically.

This is more fully discussed in . When this statement is used, no "ON" clauses are executed for the named record. It simply "re-establishes" as current, a record which had previously been read.

**READ record SORTED 'set-name' VIA field**

This statement instructs TQL to issue a DMS Sorted retrieval (DMS Format 6) against the record which belongs to 'set-name'.

The specification "field" is the name of a field which contains the key of the record to be read.

This statement will only retrieve one record. If more records are desired from the 'set-name' specified, the "READ record SET" statement can be used to proceed from the record originally retrieved with this statement.

**REMOVE record FROM 'set-name'**

This statement instructs TQL to issue a DMS REMOVE of the specified record from the specified 'set-name'.

**TQL/DMS: DISPLAY/REPORT**

As with conventional file I/O, all I/O statements used in the TQL DECLARATIVES SECTION, can also be used in the DISPLAY and REPORT DIVISION. The one additional statement that can be used in these DIVISIONs is the general READ statement of the format:

**READ record [ BY key-field ]**

This is generally referred to as the "primary read", around which the runtime TQL logic is generally driven. Although simple in format, its actual function when dealing with DMS records can be quite diverse, depending on the **LOCATION MODE** within DMS, of the record being retrieved. These variations are discussed in this section.

**READ: Indexed records**

When retrieving DMS records by Index keys, regardless of the actual DMS LOCATION MODE, the READ acts identically to a READ against a conventional Indexed MIRAM file. The initial retrieval will return the first record in the index structure, or if an actual 'key-value' is supplied at runtime, the initial retrieval will return the first record with that key value or the next highest key value. Any subsequent retrievals will continue from the first record retrieved. Runtime FROM and TO commands may be used to limit the retrievals based on low and high key values supplied by the end user.

If the DMS record is defined as "LOCATION MODE INDEXED" then a general read (without a BY) will apply to index key number one. Use the optional BY clause to designate that the read should use some other index 'key-field'.

If the DMS record is not defined as "LOCATION MODE INDEXED", and retrieval is desired by an index key that has been defined for that record, use the BY clause to designate the desired key.

Use the **runtime BY** clause to retrieve records on some index key other than key one.

The remaining interpretations of the READ command apply to records that are not of LOCATION MODE INDEXED and are not being retrieved with the BY clause (such as a CALC LOCATION MODE record being retrieved on an INDEX which has been defined for that same record).

#### READ: CALC records

When the generalized read statement is for a record whose DMS LOCATION MODE is CALC, the read reacts differently, depending on runtime requests.

If the user issues a runtime command that does not supply a record key for retrieval, then CALC records will be retrieved in the order in which they appear in their particular DMS AREA. Use function key two ( as a run-time NEXT record request), to continue to return records by area.

If the end user does supply a key, the requested record will be retrieved randomly, by the CALC record key. Also, when a key has been supplied, the use of is interpreted as a request for the NEXT record with the same CALC key (the DMS Format 5 NEXT DUPLICATE retrieval.)

In summary, with no key supplied at runtime, CALC records are retrieved by area. When a key is supplied, only those records with that key are retrieved.

#### READ: Direct and Via Set records

When the generalized read statement is for a record whose DMS LOCATION MODE is Direct or Via Set, these records will be retrieved on an area basis.

Records of this type are usually not retrieved through the "primary" TQL read, except for the purpose of doing REPORT summaries, or possibly for EXPORTing all records of a particular type to a personal computer.

### TQL/DMS: Currency Considerations

In conjunction with TQL programming, it is important to understand how TQL programs are affected by DMS record currency. Generally speaking, currency should rarely be of concern in the typical TQL/DMS program, as TQL takes steps to establish the appropriate currency required for various verbs and to maintain currency continuity from "screen to screen" when used interactively.

Some types of database structures, however, tend to introduce ambiguities into the picture, due to the fact that TQL operates within the framework of an existing, fixed "logic cycle". These problems will be discussed in this section.

With TQL DISPLAY processing, currency is cleared at the beginning of each new terminal command. Thus, the currency of one runtime command will not conflict with that of another command. However, when a single command (for example "listing" a large number of records) spans numerous output screens, TQL automatically maintains currency continuity from "screen to screen". Once again, this need not be of concern to the TQL programmer.

Another point deals with the currency which is required prior to execution of certain DMS verbs. For example, in a COBOL DMS program, if a user intends to INSERT a record into a set, he must make sure this record is CURRENT OF RUN-UNIT, or an error will occur. TQL automatically makes the required record current prior to issuing the DMS verb in question so that this is of no concern for the TQL programmer.

The major concern with currency, that TQL does not handle automatically, is one that occurs when utilizing a database structure known as a "Bill of Material" structure. This does not refer necessarily to a "Bill of Material" usually associated with Manufacturing systems. This term is used in a broader sense to describe any database structure which consists of two SET relationships defined between the same two RECORDS.

If this environment is not applicable to your particular database environment, you may ignore the rest of this section, as the remaining information is directed to the reader who is already quite familiar with "Bill of Material" structures, and their inherent DMS programming problems.

**Example:**

Let us look at an example. Specifically we will look at the Manufacturing structure from which the database term "Bill of Material" was derived:

This structure presents no problem if we want to simply "read" PART-MASTER records, and then "read" **only** the PROD-STRUCTURE records using **one** of the sets indicated. This structure does present a problem when we need to use **both sets** in a single TQL program and if we modify our intended logic so that we issue a "READ PART-MASTER OWNER 'WHERE-USE-SET' **for each read** of a PROD-STRUCTURE record.

The problem arises from the "fixed logic" environment of TQL. TQL generally assumes that a record accessed in a "primary" TQL read will not also be accessed in some later "secondary" read. This example causes **two** problems, both arising from the secondary retrieval of a PART-MASTER record:

Destruction of the desired currency within the COMPONENT-SET.

Loss of currency (or positioning) within the original PART-MASTER record retrieval issued as the "primary" read.

There is, fortunately, a solution for both problems. The following TQL program makes use of both, that are described in the text following the program.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TQLBOM 'BILL OF MATERIAL EXAMPLE'.
DATA DIVISION.
FILE SMC033.
RECORD PART-ANCHOR.
RECORD PART-MASTER.
* PART-KEY
* PART-DESCRIPTION
ENTRY VIA PART-ANCHOR 'PART-SET'
SORTED BY PART-KEY.
RECORD PROD-STRUCTURE.
* STRUCTURE-QTY
DECLARATIVES SECTION.
ON READ OF PROD-STRUCTURE
GET PART-MASTER OWNER 'WHERE-USE-SET'
READ PROD-STRUCTURE CURRENT.
DISPLAY DIVISION.
LIST: READ PART-MASTER, PART-KEY, PART-
DESCRIPTION
MORE$ 10 (READ PROD-STRUCTURE SET 'COMPONENT-
SET'
PART-KEY, PART-DESCRIPTION, STRUCTURE-QTY)
USING TQLBOM01.

```

(Commented lines under each record are used to indicate in which records the DISPLAYed data items are contained).

The program first retrieves an initial PART-MASTER record (probably from a key supplied by an end-user at runtime). It then enters a repeat loop which is intended to show the "components" of the original part. This is done by retrieving PROD-STRUCTURE records from the COMPONENT-SET. By use of a DECLARATIVE ON statement, each read of a PROD-STRUCTURE (in the MORE\$ loop) results in a retrieval of its WHERE-USE-SET owner, which is another PART-MASTER!

The OWNER retrieval is done with a **GET** statement. This is extremely important!

For the following types of TQL retrieval statements:

```

READ record OWNER 'set-name'
READ record USING field
READ record VIA field

```

the use of **GET** instead of **READ**, informs TQL that the retrieval should have no permanent effect on currency for the record involved. The GET of the PART-MASTER OWNER makes that record current at that moment, **and**, any further retrievals at that point would be based on that particular record's currency.

When the TQL program eventually returns to the "primary" READ, issued at the beginning of the DISPLAY (on line 23), TQL will automatically restore the currency for the last record retrieved with the **"primary" READ**.

You might think of the secondary GET to the PART-MASTER as having only a transient effect on its currency. Permanent currency is only established with the use of a **READ** statement. This "transient" currency feature is only supported for the three statements listed above, as these are the statements most likely to be involved in an ambiguous "Bill of Material" structure.

Even with this facility in place, we still have a problem with the program. We need to reestablish currency for the PROD-STRUCTURE record, after each retrieval of a PART-MASTER Owner. The READ CURRENT statement (coded on line 20) allows the programmer to continually restore the correct COMPONENT-SET currency that is destroyed each time a PART-MASTER Owner is retrieved (line 19.) This should be the only time that a TQL programmer should have to manually establish record currency.

As was mentioned previously, this procedure can be quite confusing unless the entire "Bill of Material" concept is well understood. Most TQL programmers will hopefully not be faced with coding which involves these types of structures. If they do exist in your system, the facilities described above should allow you to deal with them in TQL programs.

## Runtime Database Errors

DBMS errors which occur during TQL session initialization, such as not having the DBMS job running in the system, or not having the correct DMCL loaded, will simply be reported to the user with an appropriate informational message which explains the reason the TQL session can not be started.

If, during the execution of a TQL/DMS program, a database rollback occurs due to **record/area contention** (QW15 and QW18), an informational message will be displayed to explain that the session was aborted due to some type of resource contention, and must be restarted.

All other "unexpected" runtime errors are reported in a format which is intended for analysis by the TQL programmer. This information (similar to that displayed in the Unisys supplied DMS-STATUS modules) is displayed on the standard TIP screen format TF\$DMSER that is illustrated below.

```
Data Base Management System Error Has Occurred 23 JAN 99 11:06

Aborting program
Call return address
```

```
Error status
Rollback status
Error record
Error set
Error area
Last good record
Last good area
Dbkey page/line
Direct dbk page/line
```

Once displayed, this error screen will remain until some response is given from the keyboard (, , etc.), at which time the TQL session will be terminated.

The error status and rollback status codes displayed on the TF\$DMSER screen format will always be the standard DMS errors, extracted directly from the DMCA of the TQL program. Users should see the Unisys DMS documentation for the appropriate descriptions.

Examination of the LAST GOOD RECORD/AREA fields, as well as the ERROR RECORD/SET/AREA fields will normally lead the TQL programmer to the portion of the program that is causing the DMS error.

## TQLRUN - TQL Runtime Interpreter

### TQLRUN Features

The supplied transaction code **OPEN** begins the execution of a TQL program. While executing the program a user can:

- request the display of data using a pre-defined display format
- request the generation of a pre-defined report
- list selected fields (at the terminal using a free-format display)
- print selected fields (at the site printer or an auxiliary printer)
- export data to a file or printer
- apply constraints to the available data by including with a command certain conditions that must be met before data is to be displayed.
- select subsets of data for further processing
- sort selections of data in an alternate sequence
- calculate simple statistical values (e.g., total, average, minimum, etc.)

The following sections describe the initial execution of a TQL program and the various commands that are available to the TQL user.

### Function Keys

The TQL function keys are listed in the table below along with the default TIP function keys assigned to them and a brief description of their use.

The system administrator can modify the function key assignments by using the TQLMON [UC](#) command. Verify the function key assignments for your site with your administrator.

TQL Function Key	Default TIP Function Key	Description
CANCEL	MSG-WAIT	Cancel or terminate the current operation.
REFRESH	F1	Refresh the current screen.
ACK	F2	Acknowledge operations such as deleting a record.
NEXT	F2	Advance to the next record(s).
PREV	F3	Return to the previous



		record(s).
UPDATE	F4	Update the currently displayed record.
MORE	F9	Show more detail records.
DELETE	F10	Delete the currently displayed record.

---

## TQL Program Execution

The **OPEN** transaction causes the TQL interpreter to execute a TQL program. All TQL programs operate interactively to allow the user to enter commands that are processed by the TQL program.

It is also possible to catalog a TQL program as a transaction code. When this is done the program name is all that is keyed in at the TIP prompt. (See [Direct Execution of TQL Programs](#))

### Syntax:

```
OPEN [ progname [ command ] ]
```

### Where:

#### progname

Name of the TQL program to run. If a program name is not specified then a menu screen is displayed. (See Additional Considerations)

#### command

Optionally use this parameter to execute the specified command.

The inclusion of this initial command bypasses the display of the standard TQL prompt screen (See Additional Considerations) since the command is already known.

When this command is completed, the OPEN transaction will terminate normally. This feature is provided primarily to allow TQL programs (and an associated command) to be executed by an external transaction. The TQL EXECUTE command is an important command for this type of processing.

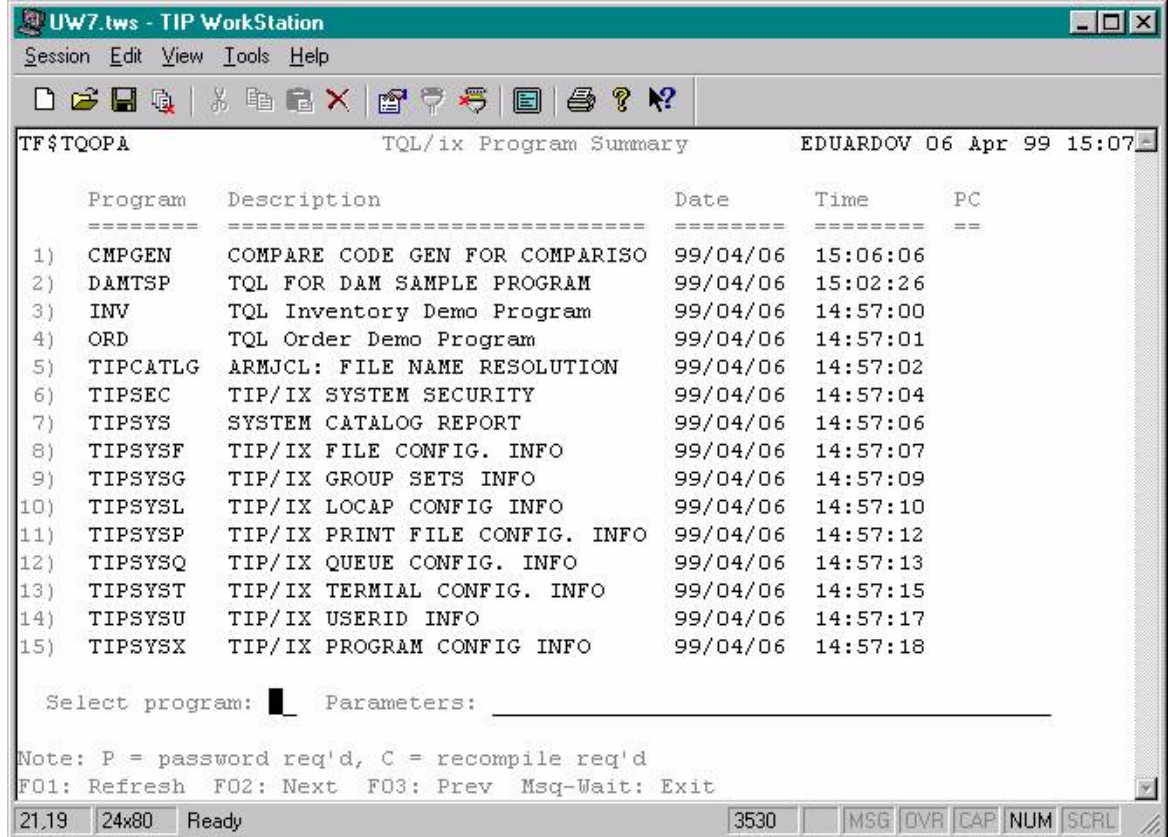
The backslash character (\) may be used to separate multiple commands entered at the command prompt (See Multiple TQL Commands ).

### Example:

```
►OPEN TQLTSP EXECUTE REPORT1
►OPEN TQLTSP EXECUTE SELECT1 \ RPT1 ON AUX1
```

## Menu Screen

As mentioned above, if the user does not supply the name of a TQL program then a menu screen similar to the following one will be displayed. The user may then enter the selection number of the desired program and press the **TRANSMIT** key, or press the **CANCEL** key to terminate the **OPEN** transaction.



```

UW7.tws - TIP WorkStation
Session Edit View Tools Help
TF$TQOPA          TQL/ix Program Summary          EDUARDOV 06 Apr 99 15:07
-----
Program  Description          Date      Time      PC
-----  -----
1)  CMPGEN  COMPARE CODE GEN FOR COMPARISO  99/04/06  15:06:06
2)  DAMTSP  TQL FOR DAM SAMPLE PROGRAM      99/04/06  15:02:26
3)  INV     TQL Inventory Demo Program      99/04/06  14:57:00
4)  ORD     TQL Order Demo Program          99/04/06  14:57:01
5)  TIPCATLG ARMJCL: FILE NAME RESOLUTION    99/04/06  14:57:02
6)  TIPSEC  TIP/IX SYSTEM SECURITY          99/04/06  14:57:04
7)  TIPSYS  SYSTEM CATALOG REPORT          99/04/06  14:57:06
8)  TIPSYSF TIP/IX FILE CONFIG. INFO        99/04/06  14:57:07
9)  TIPSYSG TIP/IX GROUP SETS INFO          99/04/06  14:57:09
10) TIPSYSL TIP/IX LOCAP CONFIG INFO        99/04/06  14:57:10
11) TIPSYSP TIP/IX PRINT FILE CONFIG. INFO  99/04/06  14:57:12
12) TIPSYSQ TIP/IX QUEUE CONFIG. INFO       99/04/06  14:57:13
13) TIPSYST TIP/IX TERMIAL CONFIG. INFO     99/04/06  14:57:15
14) TIPSYSU TIP/IX USERID INFO             99/04/06  14:57:17
15) TIPSYSX TIP/IX PROGRAM CONFIG INFO      99/04/06  14:57:18

Select program: █ Parameters: _____

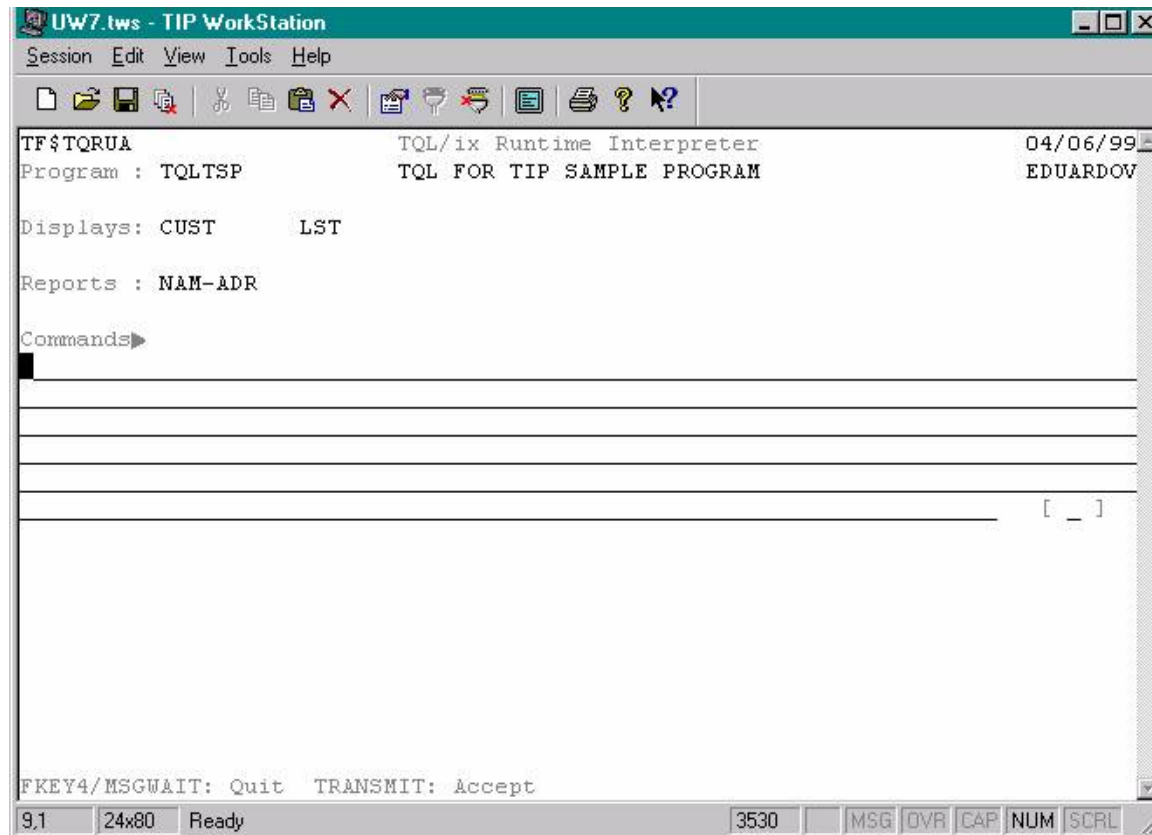
Note: P = password req'd, C = recompile req'd
F01: Refresh F02: Next F03: Prev Msq-Wait: Exit
21.19 24x80 Ready 3530 MSG OVR CAP NUM SCRL
  
```

The menu screen can be disabled by a configuration option that may be modified using the [TQLMON UC](#) command.

In the column labelled PC, a 'P' indicated that the program is password protected, a 'W' means item compiled with warnings, a 'C' means item requires recompile.

## Prompt Screen

If the program selection is valid the following prompt screen is displayed. This screen format is used to enter runtime TQL commands



## Multiple TQL Commands

You may enter multiple TQL commands by separating each command with a backslash character (\). Multiple commands may be entered into the TQL prompt screen by the end user, on the OPEN command line, or supplied in the CDA by an application program.

### Example:

```

OPEN TQLTSP PRINT NAM-ADR \ PRINT NAM-ADR
IF NO-TERM > 50
    
```

The above command executes the TQLTSP program and prints the NAM-ADR report using every record in the file. It then prints the NAM-ADR report a second time, this time only including the records for which the field NO-TERM is greater than 50. The **OPEN** transaction then terminates normally.

## AD HOC Modifiers

When a TQL command is executed the user can supply one or more AD HOC modifiers to alter the behavior of the specified command. These modifiers may constrain the data to certain ranges or conditions, or perform some other operation in addition to the displaying or reporting of data.

This section provides a reference for all the available AD HOC modifiers.

### BY

The **BY** modifier allows the user to specify an alternate key to use for the execution of a command.

If a specific key field name is not known, the reserved names **KEY1 ... KEY10** may be used to implicitly specify a key of the file.

#### Syntax:

```
BY { field-name | KEYn }
```

#### Where:

field-name The field name of a key for the file that is being accessed.

KEYn A name in the range from KEY1 to KEY10 to explicitly specify a key for the file.

#### Example:

```
PRINT REPORT1 BY CUSTOMER
PRINT REPORT2 BY KEY3
```

#### Additional Considerations:

When formatting key-values (including [FROM](#) and [TO](#) values) TQL uses the declared PICTURE clause information for a field that is specified in a **BY** clause. If you use the reserved field names **KEY1 ... KEY10**, TQL references the appropriate record layout for the first definition that applies to the specified key field.

If a group item name is used in a **BY** clause, TQL will use the definition of the elementary item or items to determine how to interpret the key data. Using the following data definition, the accompanying table shows how some sample runtime clauses would be interpreted.

```
...
05 SECOND-KEY .
   10 WAREHOUSE PIC 9(2) .
   10 AISLE PIC 9(6) .
05 K2 REDEFINES SECOND-KEY PIC 9(8) .
```

...

Statement	Interpretation
REC BY K2 FROM 12	00000012
REC BY KEY2 FROM 12	12000000
REC BY SECOND-KEY FROM 12	12000000

The formatting difference occurs if the field name specified is a redefinition of a key field (See first row of table).

## FROM

The **FROM** modifier is used to specify a lower limit for the active key.

### Syntax:

```
FROM literal ...
```

### Where:

literal Processing is to begin with the first record which has a key greater than or equal to the value of this literal. Any numeric or string literal can be used. Multiple literals may be specified to allow the user to use complex keys.

### Example:

```
DISP1 BY NAME FROM "JOHN"
DISP2 BY JOB-NO FROM 2000
DISP3 BY CUST-NO FROM "MAP" 3
```

### Additional Considerations:

When specifying a key value with the **FROM** modifier the key value may be a partial value rather than a specific value. (i.e., FROM 'JO' instead of FROM 'JOHN')

When a partial value is specified, it is padded with LOW-VALUES. This allows a **FROM** modifier to include all records whose key begins with the FROM value.

See the [BY](#) modifier in this section for an explanation of how complex keys are handled.

## GO

The **GO** modifier is used in conjunction with the [CHANGE](#) or [UPDATE](#) command to allow batch processing of multiple records without user confirmation of each change.

**Syntax:**

GO

**Additional Considerations:**

If any data in the record fails to pass any programmed [VERIFY](#) clauses, the record will be displayed for operator correction before allowing the update to continue.

The TQL program must specifically allow the use of the **GO** modifier since it can be dangerous if not used carefully.

## IF

The **IF** modifier allows the user to conditionally include records in the processing of a TQL command.

**Syntax:**

**IF** expression

**Where:****expression**

Any valid TQL expression. The expression may include arithmetic, relational and Boolean operators.

**Example:**

```
SUM INV-QTY IF LOCATION = "WAREHOUSE"  
COUNT ORD-HDR IF COUNTRY = "GERMANY" AND  
ORD-TOTAL < ORD-ESTIMATE / 2
```

**Additional Considerations:**

The **IF** modifier is used to restrict processing of any driving record reads (READ statements without [FROM](#) or VIA clauses).

The supplied expression is evaluated after any READ declaratives are executed. This allows the expression to accurately use fields in the driving record and any records read in the declarative using the VIA clause.

To apply a condition to any child or detail records, read using a [FROM](#) clause, use the [WHERE](#) modifier described in this section.

## INTO

The **INTO** modifier allows a user to specify a pathname for a UNIX file which is to receive the output of a command.

**Syntax:**

```
INTO path
```

**Where:**

**path** Any valid UNIX pathname (relative or absolute).  
The pathname may have quotes around it but they are not required.

**Example:**

```
EXPORT CR-NAME CR-PHONE CR-ORD-DATE INTO  
/export/tql/call_list  
EXPORT XR-COMPANY XR-CONTACT INTO  
"./john/contacts.txt"
```

**Additional Considerations:**

Typically TQL strings are forced to UPPER case unless the literal notation ("text" ) is used. This is not necessary for the **INTO** modifier. Whether the path is specified with the literal notation or just quoted (or not quoted at all) the text is never modified.

## key-value

The **key-value** modifier is used to specify a key value for a specific record to process.

**Syntax:**

```
literal ...
```

**Where:****literal**

The command is to process the record which has a key equal to the value of this literal. Any numeric or string literal can be used.

Multiple literals may be specified to allow the user to use complex keys.

**Example:**

```
DELETE CUST-REC "ALP00000"  
UPDATE "Q" 1994 "Y" 0
```

**Additional Considerations:**

See the [BY](#) modifier in this section for an explanation of how complex keys are handled.

## MOVE

The **MOVE** modifier is used to copy the value of an expression into one or more destination fields.

### Syntax:

```
MOVE expr TO field ...
```

### Where:

**expr** An expression to use as the source for the move operation. The expression may be a simple field or literal or may be a more complex TQL expression.

**field** The name of a field to use as the destination for the move operation. Multiple destination fields may be specified and all will receive the source value.

A **MOVE** modifier may appear before or after a command that is to be executed. If the **MOVE** modifier appears before the command it is executed once, before the TQL command is executed. If the **MOVE** modifier appears after the command it is executed every time a record is read.

Several **MOVE** modifiers may be entered. The **MOVE** modifier is useful in combination with the [GO](#) modifier to make bulk changes to a file.

The **MOVE** modifiers that are executed before the command are ideal for passing last minute information to the TQL program (usually by moving a value to a working storage field). For example, a report could examine a field in WORKING-STORAGE to determine the desired number of lines on a page or the number of columns of labels that are to be generated.

### Example:

```
MOVE 'LA' TO WS-DEPOT PRINT REPORT3 FROM  
'BOLTS' TO 'NUTS'  
MOVE TAX * 1.08 TO TAX
```

In this example, the statement **MOVE 'LA' TO WS-DEPOT** is executed once only (before the **PRINT** command begins execution).

In this case, it is presumed that some code in the TQL program (an **ON READ** clause perhaps or an **IF** statement within **REPORT3**) examines the value in the field **WS-DEPOT** to select records or other information.

The statement **MOVE TAX \* 1.08 TO TAX** is executed for every primary record that is read. Again, the presumption is that some sort of 8% surtax is being applied to value being reported from the field **TAX**.

### Additional Considerations:

**MOVE** modifiers that appear after the TQL command are executed immediately after any **DECLARATIVE (ON READ)** processing.



## ON

The **ON** modifier allows the user to select an output printer destination for a TQL command.

**Syntax:**

```
ON printer [ NO HEADING | WITH HEADING ]
```

**Where:**

**printer**

Any valid TIPPRINT destination.

The **WITH HEADING** or **NO HEADING** clauses allow the user to control whether or not the AD HOC heading page is generated.

**Example:**

```
INV-LIST ON AUX1
EXPORT NAME ADDR IF NAME BEGINS WITH "DR."
ON PRNTR NO HEADING
PRINT CUST-RPT ON HP-PRNTR
```

## SORT

The **SORT** modifier allows the user to process selected records in ascending or descending order using one or more fields as a sort key specification.

**Syntax:**

```
SORT [ default-order] [ name ]
BY field[:order] ... field[:order]
```

**Where:**

**default-order**

Specifies the default sort order.

The following values are valid for this parameter:

ASC or ASCENDING

Indicates that the default sort order is ascending.  
This is the default.

DESC or DESCENDING

Indicates that the default sort order is descending.

**name** A record, display or report name.

If a display or report is specified then the driving record is the one sorted.

If this parameter is omitted the first display in the TQL program is used to determine which record is sorted.

**field** Indicates the field or fields which are compared for the purposes of sorting. The order that the field names are specified in is significant. The fields are assumed to be specified in major to minor order. That is, the first field is the primary sort, the next field is secondary within the primary field, etc.

**order** Specifies ascending or descending order for the accompanying field.

If order is not specified the default sort order (ascending) is used.

The following values may be used for this parameter.

**A or ASC or ASCENDING**

Indicates that the default sort order is ascending.

**D or DESC or DESCENDING**

Indicates that the sort is to be in descending order.

**Example:**

```
PRINT STATE COMPANY TELEPHONE SORT BY
STATE:D COMPANY TELEPHONE:D
```

This command would sequence the currently selected records (selecting all records, if necessary) in descending order by the field STATE. Then, for each unique STATE, the companies would be put in ascending order (COMPANY within STATE) and finally, TELEPHONE numbers would be sorted in descending order for each COMPANY (TELEPHONE within COMPANY within STATE). The resulting sorted selection of records would be used to produce the output for the **PRINT** command.

**Additional Considerations:**

The **SORT** modifier may be used as a command on its own (See [SORT Records](#) ).

The real data file or files involved are not actually sorted. Instead, the high level index for the active selection (See [SELECT Subset Of A File](#) ) is reordered according to the **SORT** specifications.

If a selection is not active a selection of all records will be made in order to perform the sort.

The **SORT** compares fields in two records by doing a hardware character comparison that is not sensitive to the internal representation of the field. This implies that sorting based on fields which are binary or packed fields may lead to results which appear to be incorrect because the user may be unaware of subtle differences in internal representation (the sign of numeric fields or the byte order of binary fields, for example).

Also keep in mind that character fields which contain upper and lower case information are affected by the fact that UPPER case letters appear before LOWER case letters in the defined collating sequence (and TQL

does NOT treat upper and lower case letters as equivalent for sorting purposes).

## SUM

The **SUM** modifier is used to count and sum a field or fields to be printed after a command has completed execution.

### Example:

```
LIST CM-COMPANY FROM "A" TO "J"
      CM-NO-TERMINALS SUM CM-NO-TERMINALS
```

After the **PRINT** command has completed the following is also output.

### Additional Considerations:

The **SUM** modifier may be used as a command on its own (See [SUM Fields](#) ).

At the end of the display (or upon returning prematurely to the TQL command screen), the number of items counted will be displayed. For each field, the total, average (mean), standard deviation, maximum and minimum will be shown.

The final column (Minimum) appears only on printed reports since it is beyond the right most edge of the screen.

Only numeric fields may be specified with the **SUM** modifier.

The Deviation is the standard deviation of the data (population n-1). This value is computed according to the formula:

$$\text{SQRT} \left( \left( \text{sum of all } X * X \right) - \left( n * \text{AVG} * \text{AVG} \right) \right) / n - 1$$

Where  $n$  is the number of items,  $X$  is an individual item and  $AVE$  is the mean of all  $X$ 's.

## TO

The **TO** modifier is used to specify an upper limit for the active key.

### Syntax:

```
TO literal ...
```

### Where:

**literal** Processing is to stop with the last record which has a key less than or equal to the value of this literal. Any numeric or string literal can be used.

Multiple literals may be specified to allow the user to use complex keys.

**Example:**

```
DISP1 BY NAME TO "GEORGE"
DISP2 BY JOB-NO FROM 1000 TO 2000
DISP3 BY CUST-NO FROM "M" 3 TO "M" 7
```

**Additional Considerations:**

When specifying a key value with the **TO** modifier the key value may be a partial value rather than a specific value. (i.e., TO 'JO' instead of TO 'JOHN')

When a partial value is specified, it is padded with HIGH-VALUES. This allows a **TO** modifier to include all records whose key begins with the TO value.

See the [BY](#) modifier in this section for an explanation of how complex keys are handled.

## WHERE

The **WHERE** modifier allows the user to conditionally include records in the processing of a TQL command.

**Syntax:**

```
WHERE expression
```

**Where:**

**expression**

Any valid TQL expression. The expression may include arithmetic, relational and Boolean operators.

**Example:**

Given the following display, a runtime WHERE clause may be specified to select which transactions (TRANSACTION-REC) are to be included in the display.

```
REC: READ CUSTOMER-REC
PERFORM DISP-CUST-FLDS
DO 10 {
  READ TRANSACTION-REC FROM CUST-NO
  PERFORM DISP-TRANS-FLDS
} USING FORMAT1.
```

If the desired records were all customers (CUSTOMER-REC) with a CUSTOMER-NAME field containing 'DR' and all corresponding transactions (TRANSACTION-REC) with a TRANS-DATE field greater than 870601 then the following command would be desirable.

```
►REC IF CUSTOMER-NAME CONTAINS 'DR' WHERE TRANS-DATE > 870601
```

### Additional Considerations:

A common data organization is the presence of a parent record which contains a pointer to a variable number of child records (that often are contained in another file). The **IF** modifier applies only to fields in the parent record or controlling file read.

If a display or report also reads child or detail records (that is, records from a supplementary file) using READ FROM, the **IF** modifier can not be used to apply selection criteria to those records because the **IF** expression is evaluated before the secondary read (READ **FROM**) occurs.

The WHERE clause may be used to apply criteria to child record fields as it is evaluated after each READ FROM is executed.

The supplied expression is evaluated after any READ declaratives are executed. This allows the expression to accurately use fields in the child record and any records read in the declarative using the VIA clause.

To apply a condition to a driving record read (i.e., a READ without a FROM or VIA clause) use the **IF** modifier described in this section.

## ADHOC Commands

This section of the manual provides a reference for the TQL AD HOC commands. These commands when entered on the TQL prompt screen allow the user to execute predefined displays, print predefined reports and otherwise display and manipulate data at runtime.

When executing a command from the TQL prompt screen TQLRUN parses the supplied command in this order:

attempt to match an AD HOC command

attempt to match a display or report name  
(See [Using a Predefined Display](#) and [PRINT Predefined Report](#) )

attempt to match a saved command name  
(See [EXECUTE A Saved Command](#) )

assume the default display or report is to be executed and try matching the command text against the acceptable AD HOC modifiers (the last resort being to use the text as a [key-value](#))

## Using a Predefined Display

The TQL user may request that data be displayed according to a predefined display format. Each predefined display format has a name

which was assigned by the programmer. The display format describes which fields will be displayed and the visual format of the display.

To request a particular display, the user enters a command in the following format:

**Syntax:**

```
[display] [ ad hoc modifier ... ]
```

**Where:**

**display**

The name of the desired predefined display. A list of available display names that have been preprogrammed is displayed at the top of the TQL prompt screen.

If a name is not provided the first predefined display is executed. If no predefined displays exist then the first predefined report is executed (See PRINT Predefined Report ).

**ad hoc modifier**

Allowed AD HOC modifiers are:

```
BY  
FROM  
IF  
key-value  
MOVE  
SORT  
SUM  
TO  
WHERE
```

**Example:**

This example requests the predefined display named DISP1. Records are displayed if the field INVOICE-TOTAL has a value greater than 5,000.

```
►DISP1 IF INVOICE-TOTAL > 5000 SUM INVOICE-AMT UNIT-PRICE
```

After displaying all data screens, the total, average, standard deviation, maximum and minimum values of both INVOICE-AMT and UNIT-PRICE are displayed. The count of the number of items used to compute the average is also shown.

**Additional Considerations:**

While a display is executing several function keys are available to the user (See [Function Keys](#) ).

## ADD Record

The ADD command allows the user to add a new record (assuming that this capability is permitted by the TQL program). TQL will display a selected screen format with unprotected fields to allow the user to enter data.

### Syntax:

```
ADD [ display ] [ ad hoc modifier ]
```

### Where:

#### display

The name of a predefined display in the TQL program which will indicate the fields and screen format to display to the user. If this name is omitted, TQL will default to the first display defined in the program.

#### ad hoc modifier

The only supported AD HOC modifier for the ADD command is:

key-value

### Additional Considerations:

The screen is displayed with no initial data (unless the user supplies a [key-value](#)). The user must enter the appropriate data and press the **TRANSMIT** key. When TQL receives the data it verifies it according to any VERIFY clauses, checks for the existence of any MUST ADD fields and executes any ON ADD or ON WRITE declaratives. If errors are detected the terminal operator is notified. The terminal operator should correct the data in error and press the **TRANSMIT** key again.

## CHANGE Data

The **CHANGE** command is equivalent to the [UPDATE](#) command; it is used to update data in a file (provided that the TQL program allows that operation). See description of the [UPDATE](#) command in a separate section.

## CLOSE TQL Program

The **CLOSE** command terminates the TQL program. It is equivalent to the [END](#) command.

### Syntax:

```
CLOSE
```

## COUNT Records

The **COUNT** command counts records in the file.

**Syntax:**

```
COUNT [ name ] [ ad hoc modifier ... ]
```

**Where:**

**name** May either be a record name or a display name.  
Default: the first display defined in the program will be used.

**ad hoc modifier**

Allowed AD HOC modifiers are:

```
BY  
FROM  
IF  
SORT  
SUM  
TO  
WHERE
```

**Example:**

```
COUNT IF TIMES-RUN > 5  
SUM BASIC-CHRG TOTAL-CHRG
```

## DELETE Record

The **DELETE** command allows the user to delete a record from a file.

**Syntax:**

```
DELETE [ display ] key-value
```

**Where:**

**display**

The display format to use to display the record.

Default: the first display defined in the TQL program.

**key-value**

The specific key of the record to be deleted. This value is required and must constitute a complete (not partial) key value.

**Example:**

```
DELETE 'AEI00020'
```



**Additional considerations:**

The **DELETE** command will display a selected record and prompt the user for confirmation of the delete request - the informational message Press xxx to delete record (where xxx will be replaced with the currently defined **ACK** key, See the [Function Keys](#) section of this manual) will appear on the screen. The terminal operator should verify that the displayed record is indeed the record to be deleted.

To delete the displayed record press the **ACK** key. If any other key is pressed, TQL will not delete the displayed record and return the user to the TQL command screen.

Records cannot be deleted unless the TQL program has explicitly allowed deletes to be performed.

If the exact key of the record to be deleted is not known, the terminal operator can use the normal facilities of TQL to display records (example: display-name [FROM](#) ...). Once the correct record is displayed, pressing the **DELETE** key (See [Function Keys](#) ) will request deletion of the current record. You will then be prompted for confirmation as if you had entered the appropriate **DELETE** command.

## DROP Selection

The **DROP** command allows the user to deactivate a selection. (See the [SELECT](#) command for a description of a selection). The selection is still available for future reselection. To permanently remove a selection, use the [RELEASE](#) command.

**Syntax:**

```
DROP [ sel-item ]  
DROP ALL  
DROP TO sel-item
```

**Where:****sel-item**

Can be either a selection name or a selection number. If you specify sel-item, the selection is dropped but the current selection will not change unless sel-item is the current selection.

**ALL** Drops all selections, leaving no current selection.

**TO sel-item**

Drops all selections from the current selection up to, but not including, the selection specified by sel-item. Sel-item becomes the current selection.

**Additional Considerations:**

TQL displays the error message No Selection open! if you attempt to issue a **DROP** command without having a selection active.

Selections may be listed using the [SHOW](#) command (later in this section).

## END TQL Program

The **END** command terminates the TQL program. This command is equivalent to the [CLOSE](#) command.

**Syntax:**

```
END
```

**Additional Considerations:**

The **END** command must be spelled as shown, without abbreviation. Using the **CANCEL** function key when the TQL prompt screen is displayed is equivalent to an **END** or [CLOSE](#) command.

## ENTER Several Records

The **ENTER** command is equivalent to multiple uses of the [ADD](#) command. TQL will repeatedly display the specified display so that the terminal operator may enter new records to the file.

**Syntax:**

```
ENTER [ display ]
```

**Where:****display**

The name of the desired display to use to enter records.

Default: first display named in TQL program.

**Additional Considerations:**

The screen is displayed with no initial data. You must enter the data and press **TRANSMIT**.

To terminate the ENTER command, the terminal operator must press **CANCEL**. This causes TQL to return to the TQL command screen. The message "Record not added" signals that the ENTER operation has been completed (the last screen, at the time of CANCEL, was NOT added to the file).

When TQL receives the data it will verify it according to any [VERIFY](#) clauses, check for the existence of any MUST ADD fields and execute any ON ADD or ON WRITE declaratives. If errors are detected, the

terminal operator will be notified. The terminal operator should correct the data in error and again press **TRANSMIT** to attempt to add data.

If the display being used for the **ENTER** command has an ON ENTER clause the ON ENTER display will be executed after pressing **TRANSMIT** on the initial display. The new display will repeatedly be presented to allow the user to add child or detail records. When the user presses the **CANCEL** key the original display will again be presented to allow the opportunity to enter more records.

## EXECUTE A Saved Command

The **EXECUTE** command allows you to recall a previously saved command and immediately execute that command with no further modification. (See [RECALL a Command](#) and [SAVE a Command](#) )

### Syntax:

```
[ EXECUTE ] name [ parameters ]  
EXECUTE
```

### Where:

**name** The name of a saved command.  
If this parameter is not provided (and EXECUTE is specified), TQL presents a menu of previously saved commands for you selection.

### Additional Considerations:

The word **EXECUTE** is optional (unless you wish to view a menu of potential commands) because TQL attempts to match a name (in this order of precedence) to:

- a display name
- a report name
- a saved command for this program

Otherwise, the characters are interpreted as a [key-value](#).

## EXPORT Data

The EXPORT command is used to write data:

ON any valid TIPPRINT destination

INTO a UNIX text file

### Syntax:

```
EXPORT [ format ] [(] export-item ... [)]  
[ad hoc modifier ...]
```

**Where:**

## format

One of the following format specifiers.

Default: LOTUS 1-2-3 format (See Additional Considerations below).

ASIS Indicates that no delimiter character is wanted.

'?' Indicates that the data is to be output using the specified character as the field separator.

A string of up to eight characters may be specified within quotes.

The character string will be appended to each field that is output.

## MAPPER

Indicates that the data is to be output in a format acceptable by computer MAPPER (a tab character 'X'05' is used as a field separator).

## OFIS or SPLINK

Indicates that the data is to be output in a format acceptable by OFIS (one field per line)

TRIM Indicates that trailing spaces are to be removed from PIC X fields.

export-item One or more items to be exported (may optionally be enclosed in parentheses) separated by either commas or spaces.

The export items may simply be field names but more complex TQL expressions are allowed.

## ad hoc modifier

The AD HOC modifiers which are allowed are:

BY  
FROM  
IF  
INTO  
ON  
SORT  
SUM  
TO  
WHERE

The INTO and ON modifiers are mutually exclusive. If neither is supplied then the default destination for the EXPORT command is the TIPPRINT destination AUX0.

**Example:**

```
EXPORT CUST-NAME CUST-DATE CUST-DUE CUST-BAL * 1.08
IF CUST-COUNTRY = "CANADA" INTO "/test/export.data"
```

**Additional Considerations:**

The first non-WORKING-STORAGE field is used to determine the record for the driving READ.

The specified fields are extracted from the records based on any [IF](#), [FROM](#) or [TO](#) modifiers that are present. Each set of selected fields becomes a line in the output print file or a record in the output file.

The default export format is LOTUS 1-2-3 format and follows these rules:

Alphanumeric fields are output enclosed in single quotes.

Numeric fields are output with zero suppression and a decimal point (if appropriate).

Three spaces are output between fields.

The maximum line length for export is 2560 characters.

**Free Format LIST**

The **LIST** command may be used to list a specific set of fields (or calculated values) from records which are selected based on any AD HOC modifiers. Data is displayed for each item specified. The information is displayed in groups of four lines on the screen. Only the specified items will be displayed, nothing is displayed automatically.

**Syntax:**

```
LIST [attrib] [(] list-item ... [)] [ad hoc modifiers]
```

**Where:****attrib**

can be one or more of the following:

**WIDTH = <number>**

Number of characters per line to a maximum of 250.

Default: 80

**LENGTH = <number>**

Number of lines per page.

**SPACING = <number>**

Number of blank lines to insert between each data record.

Default: 0

**WITH [ NO ] WRAP**

Controls wrapping of the output data based on the line length.

Default: Wrapping enabled.

**list-item**

One or more items to be listed (may be enclosed in

parentheses) separated by either a comma or space.  
The list items may simply be field names but more complex TQL expressions are valid.

#### **ad hoc modifiers**

The allowed AD HOC modifiers are:

BY  
FROM  
IF  
ON  
SORT  
SUM  
TO  
WHERE

If the ON clause is not specified then the default TIPPRINT destination is AUX0.

#### **Example:**

```
LIST CUST-NAME CUST-DATE CUST-DUE
```

#### **Additional Considerations:**

The first non-WORKING-STORAGE field is used to determine the record for the driving READ.

Each screen of listed data contains a command area at the top of the screen to ask whether to continue or not.

## **Display MORE Data**

The **MORE** command causes the TQL program to proceed to the MORE\$ label in the current display. If the label MORE\$ is not encountered in the predefined display an error message will be displayed.

The MORE\$ label is usually used to specify the portion of a display that represents child information when records are organized in a parent-child relationship.

#### **Syntax:**

```
MORE
```

#### **Additional Considerations:**

The **MORE** function key (See [Function Keys](#) ) may also be used while a display is being executed to show more child or detail items.

## MOVE Field or Value

The [MOVE](#) command is documented in the AD HOC modifiers section of the manual.

## Display NEXT Screen of Data

The **NEXT** command continues displaying records from the last record displayed.

If the MORE key has been used to present subsequent screens of child record data and the [PREV](#) key has been used to move backwards in the display of child records then NEXT will move forwards through the display of child records. When the last generated screen of child records is being displayed NEXT will move to the next parent record. NEXT will only move forward through child records for previously displayed child records. MORE must be used to walk through the complete set of child records.

### Syntax:

**NEXT**

### Additional Considerations:

While a display is executing the **NEXT** key (See [Function Keys](#) ) is equivalent to the **NEXT** command.

## OPEN New Program

The **OPEN** command will terminate the current TQL program and execute the TQL program specified as a parameter to the **OPEN** command.

### Syntax:

**OPEN**[ ,d] **program-name**

### Where:

**d** This option gives you the ability to create a debug log. This log will be stored in /tmp/tqlrun.debug.

### **program-name**

The name of the (new) TQL program to execute.

## Display PREV Screen of Data

The **PREV** command returns the display to the previous record displayed.

If the MORE key has been used to present subsequent screens of child record data then PREV will move backwards through the display of child

records. When the first screen of child records is being displayed PREV will move to the previous parent record.

**Syntax:**

**PREV**

**Additional Considerations:**

While a display is executing the **PREV** key (See [Function Keys](#)) is equivalent to the **PREV** command.

## PRINT Predefined Report

The **PRINT** command produces a report that has been predefined in the TQL program.

**Syntax:**

**[PRINT] report-name [ad hoc modifier ...]**

report-name

The name of a predefined report in the TQL program.  
A list of available reports is displayed on the TQL prompt screen.

ad hoc modifier

The allowed AD HOC modifiers are:

BY  
FROM  
IF  
MOVE  
ON  
SORT  
SUM  
TO  
WHERE

If no ON modifier is specified then the default TIPPRINT destination is the printer specified in the definition of the report.

**Example:**

```
PRINT RPT1 IF COMPANY = "JOE'S GARAGE" ON AUX1
```

## Free Format PRINT

The **PRINT** command may be used to produce an ad hoc report. The user specifies which fields (or calculated values) are to be printed (instead of specifying a predefined report name).



**Syntax:**

```
PRINT [ attrib ][( ) print-item ... ( )] [ ad
hoc modifiers ]
```

**Where:**

attrib

can be one or more of the following:

**WIDTH = <number>**

Number of characters per line to a maximum of 250.

Default: 132

**LENGTH = <number>**

Number of lines per page.

**SPACING = <number>**

Number of blank lines to insert between each data record.

Default: 0

**WITH [ NO ] WRAP**

Controls wrapping of the output data based on the line length.

Default: Wrapping enabled.

print-item

One or more items to be printed (may be enclosed in parentheses) separated by either a comma or space. The print items may simply be field names but more complex TQL expressions are valid.

ad hoc modifier

The allowed AD HOC modifiers are:

```
BY
FROM
IF
ON
SORT
SUM
TO
WHERE
```

If no ON modifier is specified then the default TIPPRINT destination is PRNTR.

**Example:**

```
PRINT CUST-NAME CUST-DUE CUST-DATE ON AUX0
```

**Additional Considerations:**

The first non-WORKING-STORAGE field specified determines the record for the driving READ.

## RECALL a Command

The **RECALL** command allows you to recall a command which was previously saved using the **SAVE** command (See [SAVE a Command](#)).

### Syntax:

```
RECALL [name [parameters] ]
```

### Where:

**name** The name of a saved command.  
If omitted, TQL will display a menu of available saved commands so that the user may make a choice.

### parameters

The saved command may have utilized variables that are intended to be substituted at runtime. If this is the case, the user can enter variable parameters here (these parameters are similar to TIP command line parameters).

See the discussion and examples in the section on the "SAVE" command.

### Additional Considerations:

After a successful recall, the command will be displayed in the command area of the prompt screen. You may modify the command to suit the needs of the moment and press the **TRANSMIT** key to enter the command.

## RELEASE a Selection

The **RELEASE** command allows the user to discard a selection. (See the [SELECT](#) command for a description of a selection). The selection is no longer available for future re-selection in this session. If the selection was dropped before being released, the saved selection is removed as well. To temporarily switch to another selection, use the **DROP** command.

### Syntax:

```
RELEASE [sel-item]  
RELEASE ALL  
RELEASE TO sel-item
```

### Where:

sel-item

Can be either a selection name or a selection number.

If you specify sel-item, the selection is released but the current selection will not change unless sel-item is the current selection.

**ALL** Releases all selections, leaving no current selection.

**TO sel-item**

Releases all selections from the current selection up to, but not including, the selection specified by sel-item. Sel-item becomes the current selection.

**Additional Considerations:**

TQL displays the error message No Selection open! if you attempt to issue a **RELEASE** command without having a selection active.

Selections may be listed using the [SHOW](#) command.

## SAVE a Command

The **SAVE** command lets you save a (complex) command which can be recalled and executed later. After first using a command, to test its validity, simply insert the word **SAVE** in front of the command (which TQL conveniently redisplay on the TQL prompt screen), move the cursor after the last character to be saved and press **TRANSMIT**.

Pressing function key 8 at the TQLRUN command line is equivalent to having issued the SAVE command. In this case, TQL will preserve the exact layout of the command since the word SAVE does not have to be removed from the input.

**Syntax:**

**SAVE ...command text...**

**Where:**

...command text...

Any valid TQL command.

In addition to allowing normal TQL commands and AD HOC modifiers, the command text may include a special parameter notation that allows the user of the command to provide variable information when the command is recalled or executed.

The character string &n (where n is a digit from 0 through 8) is treated as a variable whose value may be provided at RECALL or EXECUTE time.

If a parameter has a meaningful default value, the default value may be specified by placing the default value in parentheses immediately after the parameter; example: &4(C). The default value will be used if the runtime user does not specify that parameter.

The special parameter &0 may also be used, meaning the entire parameter string as entered.

**Example:**

Assume the following command was saved under the name TEST.

```
EXPORT (CUST-NO AMT-OWING CREDIT-LIMIT)
FROM &1 TO &2 IF AMT-OWING > &3 ON &4
```

There are four variables used in this command that are expected to be supplied with values at runtime. The user (or a MENU item) could supply these parameters at runtime by using the following command.

```
RECALL TEST 100,199,1000,PRNTR
```

This would result in the display of the command shown below.

```
EXPORT (CUST-NO AMT-OWING CREDIT-LIMIT)
FROM 100 TO 199 IF AMT-OWING > 1000 ON
PRNTR
```

**Additional Considerations:**

After entering a **SAVE** command the screen format shown below is displayed with the text of the command to be saved. The user must give a name to this command to later **RECALL** or **EXECUTE** the command.

You may make last minute alterations of the text and then press **TRANSMIT** to save the command under the specified name.

If a parameter is defined within the command, but not supplied at runtime, TQL inserts the string '????' to highlight an entry that the user should fill in before using the command.

**SELECT Subset Of A File**

The **SELECT** command allows the user to select a subset of the records in a file and to have all subsequent commands operate only on the records identified by the subset. The selection is normally discarded when the user ends the TQL program, but the user may name the selection and therefore be able to retain it for later re-selection.

**Syntax:**

```
SELECT [name] ['grp/subset'] [ad hoc
modifier...]
SELECT sel-item
SELECT 'grp/subset' RENAMEs sel-item
```

**Where:**

name

May be either a record or display or report name.  
If a display or report is specified, the first record named in

the display or report defines the record type to be selected. If this name is omitted, the first display is used.

'grp/subset'

This is an optional name for the selection.

If omitted the selection will not be saved when the TQL program ends.

One or two names may be specified; if two names are specified, they must be separated by a slash.

The name (or names) must be enclosed within quotes.

If only one name is supplied, the group name will default to the user's private group.

ad hoc modifier

The allowed AD HOC modifiers are:

BY  
FROM  
IF  
SORT  
SUM  
TO  
WHERE

sel-item

Can be either a selection name or a selection number.

RENAMES

Rename a selection and make sel-item the current selection.

**Example:**

```
SELECT 'EDP/BUNCH' IF EMP-DEPT = 'MIS'
```

This command scans the entire file (since neither a [FROM](#) nor a [TO](#) modifier is specified) and selects all records where the field EMP-DEPT was equal to 'MIS'. The key values for each matching record are stored in the selection named "EDP/BUNCH". The name implies that the selection is accessible by those users in the user group EDP.

**Additional Considerations:**

A **SELECT** command with no selection criteria selects all records (if no selection with a matching *sel-item* is found).

A selection is stored as a high level index containing all key values for a given record.

When executing reports, displays, etc., performance is often much better when acting on a selection. This is because the current selection is maintained in memory allowing or faster access than disk based I/O.

No copy of the record is made. All subsequent commands will process the current information contained in the main file (with the exception of a modified key value).

A SELECT operation may be performed on an existing selection; if the second selection results in no matching records, the active selection will contain zero records and the user must issue a DROP command to return to the previous active selection.

Selections may be listed using the [SHOW](#) command (later in this section).

## SHOW Field Names and Selects

The **SHOW** command allows the user to see a list of all field names in a record, or in a predefined display or report. Alternately the **SHOW** command can be used to display all currently loaded selections (See [SELECT Subset Of A File](#) ).

### Syntax:

```
SHOW [ name ]  
SHOW SELECT
```

### Where:

#### name

The name of a record, display or report that this TQL program uses.

If this parameter is omitted, TQL displays the names of the files and records that this TQL program references.

If a predefined display or report name is given, TQL displays all of the field names that are output by that entity, along with information about the fields.

#### SELECT

Displays information about selections.

An asterisk (\*) marks the current selection. Each selection has a number that can be used as an identifier for DROP, RELEASE and SELECT commands.

### Example:

```
SHOW CUST
```

### Additional Considerations:

The information shown for each field is the name of the field, an indication of the field type and whether or not the field is a member of a key of the file (as shown in the following sample display).

The possible values for the *Type* entry are shown in this table.

Type	Description
A/E	Alpha Numeric Edited
A/N	Alpha-Numeric
ALPH	Alphabetic
BIN	Binary
BINS	Binary Signed
COND	Condition Name
FIG	Figurative Constant
FP	Floating Point
FPS	Floating Point Signed
GP	Group Item
IDN	Index Item
NE	Numeric Edited
NEF	Numeric Edited Floating Point
NEFS	Numeric Edited Floating Point Signed
NES	Numeric Edited Signed
NP	Numeric Packed
NPS	Numeric Packed Signed
NUP	Numeric Unpacked
NUPS	Numeric Unpacked Signed

The following shows some sample output from the SHOW SELECT command.

Descriptions of the displayed entries are shown in this table.

Heading	Description
#	A unique numeric identifier for the select within the session. This number may be used as a reference to the select where applicable.
Group Name	The name used to save the select when exiting from TQLRUN. The select is saved if it has not been dropped. If the selection is unnamed it will not be saved
File Record	The record selected.
Count	The number of records in the selection
Status	Can be either active or dropped. Active selects have

	an affect on any ad hoc commands.
Drop To	When this select is dropped the DROP TO select will become the active select. If the DROP TO number is 0 then no select will be active.
Type	Can be either NEW or OLD. OLD means that the select was a saved select and was loaded from disk. NEW signifies a newly created select.

## SORT Records

The **SORT** command is used to sort a selection of records by a field or fields in a chosen order.

See the [SORT](#) modifier for a description of the syntax and use of this command.

### Example:

```
SORT CUST-REC BY CM-MACHINE CM-MEMORY:D CM-NO-TERM:D
```

This command would sort the current selection for the record CUST-REC in ascending order by computer type (CM-MACHINE). Each record with the same computer type would then be in descending order by the amount of memory in the computer (CM-MEMORY). Finally, all machines of the same type and with the same amount of memory, would be in descending order by the number of terminals connected to the computer (CM-NO-TERM).

### Additional Considerations:

Use the **SORT** command rather than the **SORT** modifier when all you wish to do is sort a selection so that it can be saved in a new order for later use.

## SUM Fields

The **SUM** command allows the user to count records and total fields from within the records.

See the AD HOC modifier [SUM](#) for an explanation of syntax and use of the **SUM** command.

### Example:

```
SUM CM-NO-TERM IF COUNTRY = "CANADA"
```

### Additional Considerations:

Use the **SUM** command rather than the AD HOC modifier when all you wish to do is count records and sum fields within those records without executing some other command.



## UPDATE Record

The **UPDATE** (or alternately **CHANGE**) command allows the user to update one or more records. The current information is displayed on the screen, the user is then able to make necessary changes and press the **TRANSMIT** key.

### Syntax:

```
UPDATE [ display-name ] [ ad hoc modifier ... ]
```

### Where:

#### display-name

Specifies a display to use to update a record or a range of records.

#### ad hoc modifier

The allowed AD HOC modifiers are:

BY  
FROM  
GO  
IF  
KEY  
MOVE  
TO  
WHERE

### Examples:

Display any records where the AMOUNT-DUE field is greater than 5,000.

```
UPDATE CUST IF AMOUNT-DUE > 5000
```

The user may make any desired changes to each in turn and press **TRANSMIT**. Alternately the user can use the **CANCEL** key to abort the **UPDATE**, the [NEXT](#) key to skip to the next record or the [PREV](#) key to return to the previous record.

Display only the specified record, if it is found, which the user then may modify in the same manner as the previous example.

```
UPDATE CUST 'AEI00020'
```

The final example uses the AD HOC modifier [GO](#) to apply [MOVE](#) to all records that are selected (no user intervention is required). However, you must have an ALLOW GO statement in your program. See [ALLOW: GO \(Auto Update\)](#).

```
ALLOW GO
```

```
...
```

```
UPDATE CUST MOVE 0 TO AMOUNT-DUE GO
```

**Additional Considerations:**

Pressing the **UPDATE** key whenever TQL is used to display a record (using a predefined display) causes TQL to redisplay the same record for update. If you decide not to proceed with the update press **CANCEL** to cancel the update.

**Direct Execution of TQL Programs**

Normally you would run a TQL program, "programe", by entering one of the following from the TIP command prompt:

"OPEN programe" or "TQL programe".

To make it possible to invoke a TQL program as a transaction (without prefixing it with OPEN), you must use **smsec** to *add a program security record* for the TQL program.

You can execute **smsec** directly or start **tcm** then select "System Security".

The transaction id must be the same as the PROGRAM-ID in the TQL program. In the *Program Security Information* screen fill in "TIP\$SYS PROGRAM NAME" with "OPEN".

Once the TQL program has been defined as a TIP transaction, users can invoke it from the TIP command line with "programe".

**Calling TQL From a TIP Program**

To invoke a TQL program, a transaction program must do the following:

set CDA-PARAM(1) to the name of the TQL program,

set PIB-TRID to one of the TQL transaction codes (TQL, TQLRUN, or OPEN), then

call TIPSUB or TIPXCTL.

An application program can call TQL and pass a TQL command in the CDA. The structure of the CDA is described by the copy element TIP/TC-CDA.

If the CDA area contains unrelated data after the TQL command, you must restrict the size of the CDA passed to TQL. This is done by moving the desired CDA length to be passed into PIB-CDA-LENGTH. This is the sum of the length of the command and the length (72) of the fields that precede CDA-TEXT.

$\text{cda length} = 72 + \text{command length}$

For example, if the TQL command is 280 characters long, set PIB-CDA-LENGTH to 352 (which is 280 plus 72)

This differs from how TQL for TIP/30 handles calling from an application program. The command area passed to TQL on TIP/30 had to be exactly 238 bytes.

TQL does not pass back the contents of its CDA so the application's CDA will remain as it was before the TIPSUB to TQL.

TQL does not impose any restriction on the size of the command passed in via the CDA.

You may invoke a TQL program as a transaction by adding a program record to the TIP catalogue that defines the TQL program-id as a TIP transaction by cloning the supplied transaction named "OPEN".

Use the TIP Catalog Manager (TCM) to add the program record to the catalog. The transaction id must be the same as the PROGRAM-ID in the TQL program.

CDA Field	Description
CDA-PARAM(1)	The name of the TQL program to execute.
CDA-TEXT	An (optional) initial TQL command or commands separated by a backslash character (\).  The CDA-TEXT field is defined as 80 bytes in the copy element TIP/TC-CDA.  The program should define the CDA-TEXT field to ensure that the CDA is correctly formatted for TQL (See example which follows).

**Example:**

```

01  CDA.                                COPY TC-CDA OF TIP.
    05  TQL-CDA-TEXT                     PIC X(80) .
    05  TQL-CDA-TEXT-2                   PIC X(80) .
    05  TQL-CDA-TEXT-3                   PIC X(80) .
    ...
    MOVE 'TQL'                           TO PIB-TRID.
    MOVE SPACES                            TO CDA.
    MOVE 'ARINQ'                          TO CDA-PARAM(1) .
*
** Store CDA length to be passed (CDA header information)
** plus our extra data. 152 + 160 = 312
*
    MOVE 312                              TO PIB-CDA-LENGTH
    MOVE 'SELECT IF STATUS = 1 \'         TO TQL-CDA-TEXT.
    MOVE 'SORT DESCENDING BY CUR-BALANCE \'
                                           TO TQL-CDA-TEXT-2.
    MOVE 'REPT1 ON AUX1 '                TO TQL-CDA-TEXT-3.
    CALL 'TIPSUB' .
    IF NOT PIB-GOOD
        GO TO ERROR-CALLING-TQL.
    
```

This example appends two extra fields of 80 bytes to the predefined CDA-TEXT field and then builds a multi-line command in the 238 byte area and performs the TQL transaction.

# TQLMON - TQL Development Environment

---

## TQLMON Features

TQLMON is the environment used to develop TQL applications that are executed by the TQL Runtime Interpreter TQLRUN. TQLMON provides a set of commands for manipulating file, record and program definitions.

Commands may be entered at the command line providing any desired parameters; TQLMON will prompt for any missing information. In addition, the menu may be used to select the required function and to fill in the required information. To access the menu use the TIP 'Activate Menu Bar' key sequence as defined in the Terminal Interface section of the ***TIP Utilities Manual*** .

Access to TQLMON is required only by the TQL developers. The end user is not expected to run TQLMON at any time.

TQLMON maintains control over the elements entered into the system and it is important that the TQL system is not manipulated outside of the TQLMON environment. If the system appears to be inconsistent, see the [TQLADMIN](#) section for further assistance.

---

## TQL Development Cycle

This section describes the recommended development cycle while in TQLMON.

Step 1. Define the file to TQL.

Since TQL uses FCS to access the file it must be defined to TIP. The usual process is to define the file to TIP using the SMFILE utility and then use the AF command in TQLMON to define it to TQL.

The commands used for FILE definition are [CF](#), [NF](#), SMFILE and [UF](#), which are described later in this manual.

Step 2. Define the record definitions to describe the file contents.

It is recommended that records are defined outside of the program that will use the file. This allows for updating a record definition and having each program that uses that record automatically use the updated record when the program is compiled. The record definitions are precompiled which provides for faster compilation of programs.

The commands used for in RECORD definition are [C](#), [N](#), [NT](#), [U](#), [UT](#), [W](#) and [WT](#) which are described later in this manual.

Step 3. Define the program to manipulate data files.

When a program is compiled it reads in any precompiled records as specified. The commands involved in PROGRAM definition are [CP](#), [CPT](#), [NP](#), [NPT](#), [UP](#), [UPT](#), [WP](#) and [WPT](#), which are described later in this manual.

Step 4. Execute the program.

The program may be executed using the RUN or OPEN command while in TQLMON, by using OPEN, TQL or TQLRUN at the TIP command line or via a transfer of control from a TIP transaction to OPEN, TQL or TQLRUN.

---

## Editing

TQLMON invokes the editor defined by the catalogue entry for TQLEDT when source code editing is required. This defaults to FSE which is the most suitable editor for dealing with COBOL-style source modules. TQL maintains the source and provides a copy for editing. Changes are posted back to the system only on successful compilation. If you wish to abort your edit session but keep the changes made to date then either write the source to a location of your choice, or when in FSE, use the "E" command to save the source in an edit buffer. The next time you edit the file using FSE you will get this version back.

This saving of source will only work when the editor is FSE. Other editors such as "vi" do not know about TIP edit buffers and you will manually have to read in the saved source.

Currently, the only other supported editor for TQL is "vi". To use vi as your editor update the TQLEDT program record to point to the vi executable. If you would like support for any other editor then submit an RFC (Request For Change) to the support department for review.

---

## Compiling

To compile the source module that you are editing you must write out a copy of the source. In FSE the best command for this is the "WZ" command (write and quit). There is no need to specify an output file as you will be editing the file that TQL will be looking for. Remember that this is a copy of the original source so you can not destroy the original if you have made any mistakes.

If you are going through a repeated edit-compile cycle then the FSE command 'WE' is the best command to write and exit the editor. This command writes the current source module and then saves a copy in an

edit buffer. For larger TQL source modules (i.e. greater than 150 lines) FSE will load an edit buffer faster than a UNIX text file and on a large source module this will allow you to start editing sooner. This is only applicable if you are using FSE as your editor.

TQLMON remembers the modification time of the file before you edit the file. If the modification time has changed then the module will be compiled. If you quit from the editor without saving the changes the module will not be compiled and your changes will not be posted back into the system.

This method of invoking the compiler differs from the one used by TQL for TIP/30. Compilation was invoked by exiting from FSE and leaving an edit buffer behind (done via the E command or by hitting MsgWait). This method can not be used with TQL because an editor other than FSE might be used. The only method that can be employed is checking the modification time of the source file.

---

## Templates and Program/Record Cloning

When creating a new record or program TQLMON provides a default template as the basis for the new item. Each site may customize the default template to its own preference. The default templates are appropriately named "DEFAULT".

The commands used for template manipulation are [CPT](#), [CT](#), [NPT](#), [NT](#), [UPT](#), [UT](#), [WPT](#) and [WT](#), which are described later in this manual.

New records or programs can also be created based on an existing entity; however, templates provide some features over cloning an existing entry. When using an existing entity as the base, no special processing occurs and it is up to the programmer to change any required names such as the program identifier in the source module.

When a program or record is based on a template the template source is preprocessed to look for special text substitution markers. These are identifiers that begin with a dollar sign (\$). Currently there are three markers reserved by TQL. These are \$PROGRAM, \$FILE and \$RECORD. TQL will substitute text for these markers as required. If no substitution is appropriate or it is not a reserved marker a prompt for substitution text will be presented and the programmer can specify the text to use.

The default templates are required for the system to function correctly and therefore may not be deleted using the [DPT](#) or [DT](#) commands.

The TQL for TIP/30 method of file, record and program cloning is still supported; however, it is recommended that the template method be used where ever possible. If no template exists then is better to use the new style commands for creating new items. As an example, given a program

called TQLTSP, to create a new program based on TQLTSP use the command “NP NEWTSP TQLTSP” instead of “UP TQLTSP” and changing the PROGRAM-ID clause. This is a better method because it doesn't prevent someone from updating TQLTSP and it tells TQL that you have intentions of creating a program called NEWTSP. This prevents anyone else from creating a NEWTSP program at the same time you are.

---

## Concurrency Control

TQLMON keeps track of what modules are currently being modified. You will not be allowed to modify an entity that some one else is actively working on. You will have to defer your change until the other user is finished and has released the edit lock. The edit lock is applied and released by TQLMON only and requires no interaction with the user. The lock exists while a module is involved in an edit-compile cycle.

If for any reason TQLMON is unable to release the edit lock an invalid edit lock will remain. No one will be able to change the module even though no one is actively modifying the module. This situation may arise if the TQLMON process is terminated while the edit-compile cycle is in progress. The TQLADMIN utility has an **unlock** function for clearing this invalid edit lock. See the [TQLADMIN](#) section for details on removing invalid locks.

This section provides a summary of all the commands available in TQLMON. Each command is described in greater detail in the following section.

---

## TQLMON Command line Options

TQLMON supports a few command line options that control compile behavior. For example, when an error occurs during the compile of a record or program, the editor is invoked to display the error and provide an opportunity to fix the error. There are situations where the person doing the compile may not want to or know how to fix the problem.

The following table outlines the options and the effect produced.

Option	Description
B	When this option is specified, TQL will not invoke the editor when a compile error has occurred. This option is useful for recompiling the system without worrying about possible compiler errors. It is also a useful option when doing a large number of imports



- using CR and or CP commands.
- K This option prevents TQLMON from erasing the current contents of the log file
  - L This option produces a log file containing the errors from any failed compiles. This is usually used in conjunction with the B option.  
The log file created is \$HOME/log.COMP
  - O This option causes TQLMON to automatically overwrite entries that already exist in the TQL system. Without this option the system will ask for confirmation to replace the existing entry.  
This option is generally used when importing a large number of items that may or may not already exist. The script that loads the Inglenet supplied TQL items (\$TIPROOT/src/tql/LOADTQL) uses this option to automatically update the existing entries with a new release.

---

## Command Summary

Command	Description
AF	Define a file to the TQL system using information in the TIP catalogue if available.
AS	Add a DMS schema definition to TQL.
C	Compile a record definition from a UNIX file.
CD	Changes current working directory.
CF	Compile a file definition from a UNIX file.
COMP	Recompile records and/or programs in the TQL system without editing.
CP	Compile a program definition from a UNIX file.
CPT	Import a program template from a UNIX file.
CT	Import a record template from a UNIX file.
DEL	Delete files and/or record definition.
DP	Delete program definition.
DPT	Delete program template.
DS	Delete a schema definition.
DT	Delete record template.

E	End TQL monitor.
EDIT	Invokes TQLEDT on the specified file.
H	Display TQLMON command help information.
L	List file and/or record compilation listing.
LP	List program compilation listing.
LPT	List program template source.
LT	List record template source.
M	Generate screen format(s) for a program.
N	Create a new record definition.
NF	Create a new file definition.
NP	Create a new program definition.
NPT	Create a new program template.
NT	Create a new record template.
O	Execute (run) a TQL program.
P	Print file or record compilation listing.
PP	Print program compilation listing.
PPT	Print program template listing.
PT	Print record compilation listing.
R	Execute (run) a TQL program.
S	Display summary of files and records.
SMFILE	Invoke TIP SMFILE utility.
SP	Display summary of programs.
SPT	Display summary of program templates.
SS	Display summary of DMS schemas.
ST	Display summary of record templates.
TFD	Invoke TFD to manipulate a screen definition.
U	Update a record definition.
UC	Update TQL control record.
UF	Update a file definition.
UP	Update a program definition.
UPT	Update a program template definition.
UT	Update a record template definition.
W	Write record source to an external file.
WP	Write program source to an external file.

WPT	Write program template source to an external file.
WT	Write record template source to an external file.
XF	Print cross reference of file use.
XFC	Recompile all programs that reference specified files.
XP	Print cross reference of programs showing files and records used.
XR	Print cross reference of record use.
XRC	Recompile all programs that reference specified records.

---

## TQL Commands

### Command Summary

Command	Description
AF	Define a file to the TQL system using information in the TIP catalogue if available.
AS	Add a DMS schema definition to TQL.
C	Compile a record definition from a UNIX file.
CD	Changes current working directory.
CF	Compile a file definition from a UNIX file.
COMP	Recompile records and/or programs in the TQL system without editing.
CP	Compile a program definition from a UNIX file.
CPT	Import a program template from a UNIX file.
CT	Import a record template from a UNIX file.
DEL	Delete files and/or record definition.
DP	Delete program definition.
DPT	Delete program template.
DS	Delete a schema definition.
DT	Delete record template.
E	End TQL monitor.
EDIT	Invokes TQLEDT on the specified file.

H	Display TQLMON command help information.
L	List file and/or record compilation listing.
LP	List program compilation listing.
LPT	List program template source.
LT	List record template source.
M	Generate screen format(s) for a program.
N	Create a new record definition.
NF	Create a new file definition.
NP	Create a new program definition.
NPT	Create a new program template.
NT	Create a new record template.
O	Execute (run) a TQL program.
P	Print file or record compilation listing.
PP	Print program compilation listing.
PPT	Print program template listing.
PT	Print record compilation listing.
R	Execute (run) a TQL program.
S	Display summary of files and records.
SMFILE	Invoke TIP SMFILE utility.
SP	Display summary of programs.
SPT	Display summary of program templates.
SS	Display summary of DMS schemas.
ST	Display summary of record templates.
TFD	Invoke TFD to manipulate a screen definition.
U	Update a record definition.
UC	Update TQL control record.
UF	Update a file definition.
UP	Update a program definition.
UPT	Update a program template definition.
UT	Update a record template definition.
W	Write record source to an external file.
WP	Write program source to an external file.
WPT	Write program template source to an external file.
WT	Write record template source to an external file.

XF	Print cross reference of file use.
XFC	Recompile all programs that reference specified files.
XP	Print cross reference of programs showing files and records used.
XR	Print cross reference of record use.
XRC	Recompile all programs that reference specified records.

## AF - Add File

The AF command is used to import a file definition into the TQL system. The file must already be defined to TIP.

The file specified must match a file entry defined in the TIP security file.

The AF command has two modes. The first is single entry mode where a specific file is imported. If the file is already defined to TQL an error will occur.

The second mode is used for multiple file imports. This mode can be entered by specifying the "\*" wild-card character in the file name. In this mode files that are already defined to TQL will be updated automatically (no error will occur). This is similar behavior to the [UF](#) command.

To define a file to TQL that is not already defined to TIP you can use the [NF](#) or SMFILE command to define the file to TIP.

### Syntax:

```
AF file
```

### Where:

**file** The file name of the file to import into TQL.

### Example:

```
AF TSPFILE *> Import the file definition for TSPFILE
AF PAY* *> Import all file definitions starting with PAY
```

## AS - Add Schema

The AS command is used to add a schema definition to the TQL system. The schema must already have been created using the TIP/dbi schema compiler.

### Syntax:

```
AS schema
```

**Where:**

**schema**      The name of the schema to define to TQL.

**Example:**

```
AS DMSSCH *> Import the schema definition for DMSSCH
```

## C, CF, CP, CPT, CT - Compile From External Source

These commands are used to define items to the TQL system using source files external to the system. The item being defined to the system depends on the following commands:

C Record definition

CF File definition

CP Program definition

CPT Program template definition

CT Record template definition

The CPT and CT commands do not involve any compilation and just copy the source into the system.

When an item is compiled, TQL creates a listing that can be viewed using one of the TQLMON list commands (L or LP). If there are any compile warnings, TQL will display a message to indicate that the item was compiled with warnings and the actual warning messages can be viewed at the front of the listing.

If errors are encountered when using the C, CF or CP commands, an error list will be displayed and the editor will be invoked to allow correcting the error.

If the item being compiled already has a definition in the TQL system then a prompt for overwrite permission will be presented.

**Syntax:**

```
C file
CF file
CP file
CPT file
CT file
```

**Where:**

**file**      The UNIX file where the source is located. The file name specified is case sensitive.  
If the file name is not an absolute path name then TQLMON will look in the following directories in the following order:

- the current directory
- the directory specified by the \$HOME environment variable
- the directory \$TIPROOT/src/tql

**Example:**

```
C /u/george/src/foo.fil *> explicit file reference
(record)
C TQLTSP.trd *> look in search path (record)
CPT DEFAULT.tpd *> look in search path (program
template)
```

**Additional Considerations:**

Previous releases of TQL and TQL for TIP/30 supported compiling file definitions from external source using the C command. This functionality has been removed and is now accomplished using the CF command.

## CD - Change Current Working Directory

The CD command changes the current working directory for the current TQLMON session. This command is useful when compiling or writing modules that are external to TQL.

**Syntax:**

```
CD directory
```

**Example:**

```
CD /var/tmp
```

## COMP - Compile Existing Records and/or Programs

The COMP command will recompile records and/or programs without invoking the editor first. If any errors are encountered the editor will be invoked to provide an opportunity to fix the error before continuing. This command is most often used when a record definition (or file definition) has changed and programs that use the record have to be recompiled.

If one parameter is specified COMP assumes that program definitions are to be recompiled. Two parameters specify that the record definitions are to be recompiled. If COMP is invoked without parameters a prompt will appear to query the user whether to recompile the records, programs or both.

File definitions do not need to be recompiled.

**Syntax:**

```
COMP progname
COMP file record
```

**Where:****programe**

Name of the existing TQL program to be compiled.

**file rename**

The COMP command can be used to recompile an existing record definition. The filename and record name must be specified.

If no parameters are specified prompt will be presented to request the compilation type. The choices here are to recompile everything, records or programs.

**Example:**

```
COMP TQLTSP *> recompile the program TQLTSP
COMP * * *> recompile all records
COMP B* *> recompile all programs beginning with B
```

## DEL - Delete File or Record

The DEL command deletes a single record or single file definition from the TQL system. A file definition can not be deleted until all associated record definitions are deleted.

**Syntax:**

```
DEL file [ record ]
```

**Where:**

**file** The name of a file defined in the TQL system.

**record**

The name of a record of that file.

If this parameter is omitted, the specified file definition is deleted.

**Example:**

```
DEL PAYMST PAYREC *> delete the record PAYMST/PAYREC
DEL PAYMST *> delete the file PAYMST
```

## DP - Delete Program

The DP command will delete a program from the TQL system. All traces of the program will be removed from the system. Because of the destructive nature of this command a prompt must be answered to finalize the operation.

**Syntax:**

```
DP programe
```



**Where:****progname**

The name of a program defined in the TQL system.

**Example:**

```
DP PAYINQ *> delete the program PAYINQ
```

## DPT, DT - Delete Program or Record Template

The DPT command deletes the source for a program template and DT deletes the source for a record template. The program and record template DEFAULT is special to TQL and may not be deleted. . Because of the destructive nature of this command a prompt must be answered to finalize the operation.

**Syntax:**

```
DPT name
```

```
DT name
```

**Where:**

**name** The name of a template defined in the TQL system.  
The type of template deleted is based on the command type.

**Example:**

```
DPT TMPL *> delete the program template TMPL  
DT RECTMPL *> delete the record template RECTMPL
```

## DS - Delete Schema

The DS command will delete a schema from the TQL system. All traces of the schema will be removed from the TQL system only. This command will not affect usage of the specified schema by programs other than TQL.

**Syntax:**

```
DS schema
```

**Where:****schema**

The name of a schema defined in the TQL system.

**Example:**

```
DS DMSSCH *> delete the schema DMSSCH
```

## E - End TQLMON Program

The End command terminates interaction with the TQL monitor program and return to the calling program or the TIP command line.

**Syntax:**

```
E
```

## EDIT - Edit a Source File

The EDIT command will invoke TQLEDT on a specified file. This command is intended for editing of general text files that are not related to the TQL system. TQL will automatically invoke an editor when creating or updating TQL entries. No compilation will occur when changes are made using this command.

**Syntax:**

```
EDIT filename
```

**Example:**

```
EDIT /home/tipixusr/log.tipix
```

## HELP - Display Help Information

The help command will display help information for the TQL monitor program. Two forms of help are available. If the help command is entered without a parameter then a command summary is displayed showing the syntax for each command. If help is specified for a given command then a more detailed description of that command is displayed

**Syntax:**

```
Help [ command ]
```

**Where:**

**command**

Display help for this command.

If this is omitted then the command summary will be displayed.

**Example:**

```
HELP *> display command summary  
H AF *> display help on the AF command
```

## L - List File/Record

The LIST command will display the compilation listing of a file or record definition. This display is on AUX0. To print to another destination, see the P command.

### Syntax:

```
L file [ record ]
```

### Where:

**file** The name of a file defined in the TQL control file.

**record**

The name of a record defined for the named file.

If this parameter is omitted then the compilation listing for the named file will be displayed.

### Example:

```
L PAYMST PAYREC *> list the compilation PAYMST/PAYREC
L PAYMST *> list the file compilation listing for PAYMST
```

## LP - List Program

The LP command displays the compilation output for a program. The output is on AUX0. To print to another destination, see the PP command.

### Syntax:

```
LP name
```

### Where:

**name**

The name of a program defined in the TQL system.

### Example:

```
LP PAYINQ *> display the program PAYINQ
```

## LPT, LT - List Program or Record Template

The LPT command displays the source for a program template and LT displays the source for a record template. The output is printed on AUX0. To print to another destination, see the PPT/PT command.

### Syntax:

```
LPT name
LT name
```

**Where:**

**name** The name of a template defined in the TQL system.  
The type of template listed is based on the command type.

**Example:**

```
LPT DEFAULT *> display the DEFAULT program template
LT DEFAULT *> display the DEFAULT record template
```

## M - Make Screen Formats

This command directs TQL to generate screen formats for the indicated display definitions in a named TQL program. The generated screen formats have names as specified in the USING clause in the [DISPLAY DIVISION](#) of the TQL program.

The screen generated is a basic screen. It is intended to be an automated quick start to allow testing of a program without having to spend a lot of time to design a screen. Once the program is working as intended, you can use the TFD utility to improve the look of the screen format. While in TQLMON you can edit the screen format by using the TFD command which invokes the TIP TFD utility.

If a screen format already exists a prompt is presented for overwrite permission. This command should be used with care because it is easy to destroy a custom screen format.

**Syntax:**

```
M name [ display ]
```

**Where:**

**name** The name of a program defined in the TQL system.

The program name may be specified using standard prefix notation.

**display** The name of the TQL display definition that describes the screen format to be created. This is not the name specified in the USING clause in the DISPLAY DIVISION.

This parameter may be specified using standard prefix notation.

If this parameter is omitted, ALL screen formats will be built for the specified program. TQL prompts for confirmation that this is what you want to do.

**Example:**

Given the following:

```
PROGRAM-ID. MYTEST.
```

```
...  
DISPLAY DIVISION.  
MY-DISP: READ ...  
...  
  
USING S1.  
  
MY-DISP2: READ ...  
...  
  
USING S2.  
  
DISP1: READ ...  
...  
  
USING S3.
```

```
M MYTEST *> make all formats (S1, S2 and S3)
```

```
M MYTEST MY-DISP2 *> make the format used  
*> by MY-DISP2 (S2)
```

```
M MYTEST MY* *> make formats for displays beginning with  
*> MY (S1 and S2)
```

## N - Define New Record

This N command creates a new record definition.

The initial contents of the source file will depend on the parameters supplied. The parameters are described in greater detail in the Syntax section.

If the item already exists you will be prompted for update intentions. This is similar to using the U command.

When the user ends the editing session, TQLMON will automatically compile the record definition composed by the user. On successful compilation the source will be posted into the system. If any compile errors are encountered the source will not be posted. If you choose not to fix the errors the item will not be defined to the system and you will have to use the N command again.

### Syntax:

```
N file record [ template ]  
N file record file2 record2
```

**Where:**

**file** Name of the file that this record will be associated with.

TQLMON will substitute this text for the \$FILE marker in the source used as the template for this record.

If this first parameter is omitted a query screen will appear.

**record** The name of the record.

TQLMON will substitute this text for the \$RECORD marker in the source used as the template for this record.

**template** If there are three parameters then the third parameter is assumed to be a record template already defined to the system. This template will be used as the basis for the new record. The template DEFAULT will be used if only two parameters are supplied.

**file2 record2** If there are four parameters then the new record definition will be based on the source for the record record2 associated with the file file2.

**Example:**

```
N PAYROLL PAYREC *> create a record PAYREC associated
*> with the file PAYROLL. Base the initial source
*> on the template DEFAULT

N PAYROLL PAYREC TMPL1 *> as above but using the
*> template TMPL1

N PAYROLL PAYREC PAYMST REC1 *> as above but basing the
*> initial source on the existing record definition
*> PAYMST/REC1.
```

**NF - Define New File**

The NF command invokes the SMFILE command. See [SMFILE - Invoke SMFILE](#) for more information. This allows definition of a new file to TIP.

Once the file has been defined to TIP the AF command must be used to import the definition into TQL. The NF command is currently unable to do this step on your behalf.

## NP - Define New Program

This NP command creates a new program definition. The initial contents of the source file will depend on the parameters supplied. The parameters are described in greater detail in the Syntax section.

If the item already exists you will be prompted for update intentions. This is similar to using the [UP](#) command.

When the user ends the editing session, TQLMON will automatically compile the program definition composed by the user. On successful compilation the source will be posted into the system. If any compile errors are encountered the source will not be posted. If you choose not to fix the errors the item will not be defined to the system and you will have to use the NP command again.

### Syntax:

```
NP progname [ template ]
```

### Where:

#### progname

The name of the TQL program that is to be created.

TQLMON will substitute this name for the \$PROGRAM marker in the source used as the template for this program.

#### template

This parameter specifies the item to be used as the basis for the new program. If this parameter is omitted the new program will be based on the DEFAULT template. This name must reference either a defined template or defined program.

TQLMON will check this name against the template list first and then against the program list if no template was found.

### Example:

Given the following:

- - a template named TMPL
- - a template named BOTH
- - a program named TQLTSP
- - a program named BOTH

```
NP TESTPRO *> new program based on template DEFAULT
NP TESTPRO TMPL *> new program based on template TMPL
NP TESTPRO TQLTSP *> new program based on program TQLTSP
NP TESTPRO BOTH *> new program based on template BOTH
NP *> present query screen for input
```

## NPT - Define Program Template

This NPT command creates a new program template definition. The initial contents of the source file will depend on the parameters supplied. The parameters are described in greater detail in the Syntax section.

If the item already exists you will be prompted for update intentions. This is similar to using the UPT command.

When the user ends the editing session, TQLMON will post the template into the system

### Syntax:

```
NPT name [ template ]
```

### Where:

**name** The name of the template to define to the TQL system.

### template

Name of the item to use as the base of this template.

TQLMON will check this name against the program template list first and then against the program list if no template was found.

If no template is specified DEFAULT will be used.

### Example:

Given the following:

- - a template named TMPL
- - a template named BOTH
- - a program named TQLTSP
- - a program named BOTH

```
NPT TESTPRO *> new program template TESTPRO based on
*> the template DEFAULT
```

```
NPT TESTPRO TMPL *> new program template TESTPRO based
*> on the template TMPL
```

```
NPT TESTPRO TQLTSP *> new program template TESTPRO based
*> on the program TQLTSP
```

```
NPT TESTPRO BOTH *> new program template TESTPRO based
*> on the template BOTH
```

```
NPT *> present query screen for input
```



## NT - Define Record Template

This NT command creates a new record template definition. The initial contents of the source file will depend on the parameters supplied. The parameters are described in greater detail in the Syntax section.

If the item already exists you will be prompted for update intentions. This is similar to using the UT command.

When the user ends the editing session, TQLMON will post the template into the system.

### Syntax:

```
NT name [ template ]
NT name [ file record ]
```

### Where:

**name** The name of the template to define to the TQL system.

#### template

Name of the item to use as the base for this template. TQLMON will check this name against the record template list. If no template is specified DEFAULT will be used.

#### file record

Name of an existing record definition to use as the base for this template.

### Example:

```
NT TESTPRO *> new record template TESTPRO based on
*> the template DEFAULT

NT TESTPRO TMPL *> new record template TESTPRO based on
*> the template TMPL

NT TESTPRO FILE1 REC1 *> new record template TESTPRO
*> based on the record REC1 associated with the file
*> FILE1

NT *> present query screen for input
```

## O - Open Program

The OPEN command allows the programmer to execute a TQL program. This command eliminates the need to end the TQL monitor and then use the OPEN transaction.

When the TQL program completes, control will return to TQLMON.

### Syntax:

```
OPEN progname
```

**Where:****progname**

The name of a TQL program defined in the TQL system.

**Example:**

```
OPEN PARTINQ *> run the program PARTINQ
```

## P - Print File/Record

The P command prints the compilation listing of a file or record definition. The output may be printed on any device supported by TIPPRINT. The default print destination is PRNTR.

**Syntax:**

```
P file [,record] [,dest]
```

**Where:**

**file** The name of a file defined in the TQL system.

**record**

The name of the record to be printed.

If record name is omitted, only the file compilation is printed.

**dest** The desired print destination.

**Example:**

```
P TSPFILE *> print file listing for TSPFILE to TIPPRINT
*> destination PRNTR.
```

```
P TSPFILE,,MYPRNTR *> as above but use MYPRNTR as the
*> output device.
```

```
P TSPFILE TQLTSPR *> print the record TQLTSPR for the
*> file TSPFILE on the default printer destination.
```

```
P TSPFILE TQLTSPR MYPRNTR *> as above but use MYPRNTR as
*> the output device.
```

## PP - Print Program

The PP command prints the compilation listing of a TQL program. The output may be printed on any valid TIPPRINT destination. The default destination is PRNTR.

**Syntax:**

```
PP progname [ dest ]
```

**Where:****program**

The name of a program defined in the TQL system.

**dest** The desired print destination.

**Example:**

```
PP TQLTSP *> print program listing for TQLTSP using the
*> default TIPPRINT destination.
```

```
PP TQLTSP MYPRNTR *> print the program TQLTSP using
*> MYPRNTR as the output destination.
```

## PPT, PT - Print Program or Record Template

The PPT command prints the source for a program template and PT prints the source for a record template. This default destination is PRNTR.

**Syntax:**

```
PPT name [ dest ]
PT name [ dest ]
```

**Where:**

**name** The name of a template defined in the TQL system.  
The type of template listed is based on the command type.

**dest** The desired print destination.

**Example:**

```
PPT DEFAULT *> display the DEFAULT program template
PT DEFAULT *> display the DEFAULT record template
PPT TMPL AUX1 *> print program template TMPL on AUX1
```

## RUN, R - Run TQL Program

The RUN command allows the programmer to execute a TQL program without exiting from TQLMON. This command is identical to the "OPEN" command described earlier.

**Syntax:**

```
RUN progname
R progname
```

**Where:****progname**

The name of a TQL program defined in the TQL system.

**Example:**

```
RUN PARTINQ *> run the program PARTINQ
```

**S - Summarize File/Record**

The S command will display a list of existing file and/or record definitions that are presently in the TQL system. File and/or record names may be selected by prefix. The listing may be interrupted by pressing the **CANCEL** key.

If no output destination is specified output is sent to AUX0. The destination specified may be any valid TIPPRINT destination.

**Syntax:**

```
S [ file [record] [dest] ]
```

**Where:**

**file** The name of a file defined in the TQL system. The file name may be specified using standard prefix notation.

The file name is optional; default is all files (\*).

**record**

The record name to be listed. The record name may be specified using standard prefix notation.

If omitted only file entries will be listed.

**dest** Desired output destination.

**Example:**

```
S *> show all files
S * * *> show all files and records
S * B* *> show all files and any records starting with B
S * * PRNTR *> print all files and records to PRNTR
```

**SMFILE - Invoke SMFILE**

This command provides access to the TIP file definition utility SMFILE. This allows for defining a file to TIP without having to leave TQLMON. See the *TIP Utilities manual* for further information on SMFILE.

**Syntax:**

```
SMFILE [ command filename ]
```

**Where:****command**

A valid SMFILE command

**filename**

The name of the file that SMFILE is to manipulate.

**Example:**

```
SMFILE AD TESTFILE *> invoke the SMFILE utility to
*> define the file TESTFILE to TIP
```

```
SMFILE *> invoke SMFILE
```

## SP - Summarize Programs

The SP command displays a list of programs that are presently in the TQL system. The listing may be interrupted by pressing **CANCEL**.

**Syntax:**

```
SP [ program [dest] ]
```

**Where:****program**

The name of a program defined in the TQL system. The file name may be specified using standard prefix notation.

The program name is optional; default is all programs (\*).

**dest** Desired output destination.

**Example:**

```
SP *> show all programs
SP * *> show all programs
SP B* *> show all programs starting with B
SP B* PRNTR *> print all programs starting
*> with B to PRNTR
```

## SPT, ST - Summarize Program or Record Templates

The SPT command displays a list of program templates that are presently in the TQL system. The ST command displays a list of the record templates that are presently in the system. The listing may be interrupted by pressing **CANCEL**.

**Syntax:**

```
SPT [ template [dest] ]
```

```
ST [ template [ dest ] ]
```

**Where:**

**template** The name of a template defined in the TQL system. The template name may be specified using standard prefix

notation.

The template name is optional; default is all templates (\*).

The template type depends on the command used.

**dest** Desired output destination. This may be any valid TIPPRINT destination.

**Example:**

```
SPT *> show listing of all program templates
ST *> show listing of all record templates

SPT B* *> show listing of all programs templates
*> starting with B

SP B* PRNTR *> print listing of all record templates
*> starting with B to PRNTR
```

## SS - Summarize Schemas

The SS command displays a list of schemas that are defined in the TQL system. The listing may be interrupted by pressing **CANCEL**.

**Syntax:**

```
SS [ schema [dest] ]
```

**Where:**

**schema**

The name of a schema defined in the TQL system. The file name may be specified using standard prefix notation.

The program name is optional; default is all schemas (\*).

**dest** Desired output destination.

**Example:**

```
SS *> show all schemas
SS * *> show all schemas
SS B* *> show all schemas starting with B
SS B* PRNTR *> print all schemas starting
*> with B to PRNTR
```

## TFD - Invoke TFD

The TFD command provides access to the TIP screen definition utility TFD. This allows for screen editing without having to leave TQLMON. See the *TIP Utilities manual* for further information on TFD.

When TFD is invoked from the TIP command line or a UNIX shell, any special characters must be escaped to avoid special shell processing.

When in TQLMON this is not required because TQLMON will take care of any required name conversion for you.

**Syntax:**

```
TFD [ screen ]
```

**Where:****screen**

Name of the screen to edit. If no parameter is supplied with the command TFD will start without presenting a screen.

**Example:**

```
TFD MY$SCREEN *> edit the screen MY$SCREEN
*> (no need to escape the "$").
```

## U - Update Record Definition

This U command updates an existing record definition.

When the user ends the editing session, TQLMON will automatically compile the record definition composed by the user. On successful compilation the source will be posted into the system. If any compile errors are encountered the source will not be posted.

**Syntax:**

```
U file record
```

**Where:**

**file** Name of the file that this record is associated with.

**record**

The name of the record to update.

**Example:**

```
U PAYROLL PAYREC *> update the record
*> PAYREC associated with the file PAYROLL.
```

## UC - Update Control Record

The UC command is used to perform maintenance operations on the control record in the TQL control file. The control record contains configuration selections that govern certain aspects of the behavior of TQL.

**Syntax:**

```
UC
```

There are no parameters for this command.

The TQLMON UC command displays the screen format shown below.

Pressing the **CANCEL** key exits the “UC” command without changing the control record.

```

TQL/ix Integrated Development Environment: 1
TF$TQLCA      TIP/ix Query Language - Control Record Maintenance      09 Apr 99

  Created: 99/04/05 14:24:53          Read Password: _____
 Modified: 99/04/05 14:25:21          Write Password: _____

TQL Path: /home1/tipix23/tql
_____

      Maximum Reads: 1000          Use TIP/ix Currency: N          FUNCTION KEYS
OPEN Security Level: 255          Use TIP/ix Decimal: N          =====
      Display TQL Menu: Y          Use Micro Focus Sign: N          MSG_WAIT: 0
      Use Paging File: N          Emulate TIP/30 TQL Math: N          Refresh: 1
      Journal Program: Y          Always TREN Updates: N          Next: 2
      Print Title Page: N          Allow Reset READ FROM: N          Previous: 3
      FIELDS OF Group: N          Preprocess Commands: N          Update: 4
      READ VIA ID Bypass: N          READ FROM Check Bypass: N          More: 9
      Ambiguity Resolution: N          Failed Read Initializes: N          Delete: 10
      Declarative READ FROM: N          Use DOS extension: N          Ack. Del.: 2

```

### Where:

#### Read Password

The user may enter a value in this field to set a new read password for the TQL control file.

TQL programmers wishing to perform read operations on the TQL control file are required to supply this password.

If a password is set this field will be shown as all asterisks.

#### Write Password

The user may enter a value in this field to set a new update password for the TQL control file.

TQL programmers wishing to make changes to the contents of the TQL control file are required to supply this password.

If a password is set this field will be shown as all asterisks.

#### TQL Path:

The path defining where the TQL system is located.

#### Maximum reads

This configuration option controls the default value to be



used as the MAXREAD specification if a TQL program does not specify the clause.

This value specifies the maximum number of reads to be performed before relinquishing temporary control to TIP via a TIPTIMER call. TQL checks for user interruption every MAXREAD number of records read.

Default: 1000

#### **Use TIP Currency**

When this is set to Y, TQL will use the currency symbol defined to TIP instead of the default currency symbol "\$".

Default: N

#### **OPEN Security Level**

This configuration option controls the access to the OPEN command in TQLRUN. The user's security value must be equal to or less than this value to allow use of the OPEN command.

If this value is 0 then no one can run another TQL program from a TQL program.

Default: 255

#### **Use TIP Decimal**

When this is set to Y, TQL will assume DECIMAL-POINT IS COMMA if the decimal point defined to TIP is not the "." character.

Default: N

#### **Display TQL MENU**

This configuration option controls whether TQL is to display a menu of available TQL programs if TQL is invoked without a program name (for example: a user executes the transaction "OPEN" with no parameters).

Default: Y

#### **Use paging file**

Not used.

#### **Journal TQL program start**

Not used.

#### **Print title page**

A block-letter style header page will be produced if Y is specified.

Default: N

**FIELDS OF Group**

When this option is set to Y then any group item used in a display will be replaced by all non-FILLER subfields of the group.

Default: N

**READ VIA ID Bypass**

When this option is set to Y then a READ VIA will not validate the record retrieved using the ID clause for that record.

**Note:** Use of this option is highly discouraged. It is provided to support TQL programs that took advantage of a bug in TQL for TIP/30 in which ID clauses were ignored on a READ VIA statement. The offending programs should be corrected and this flag reset to enforce the ID clause validation.

Default: N

**Ambiguity Resolution**

When this option is set to Y then TQL will not report ambiguous references in ad hoc statements when the reference is a full data name. In this case TQL will use the first field that matches. TQL will continue to flag errors on ambiguous substring abbreviations.

Default: N

**Declarative READ FROM**

When this option is set to Y the behavior of the READ FROM statement in a declarative is changed to always issue a SETL. When the option is N the SETL is only done for the first READ FROM execution for a given parent record.

Default: N

**Use Micro Focus Sign**

Defines which compiler's numeric unpacked sign convention to use. If using the Micro Focus COBOL compiler, set this to Y.

Default: N

**Emulate TIP/30 TQL Math**

TIP/30 TQL always added 3 digits of precision to the left operand of a divide operator in a compute statement. In addition, all math results and moves involving numerics were done with rounding.

When this flag is set to Y TQL will emulate this behavior. When the flag is set to N TQL does not do automatic

rounding and the intermediate results for numeric operations follows those for OS/3 COBOL '85.

Default: N

### **Always TREN Updates**

This flag controls whether TREN is called for each record in an UPDATE or only on completion of the UPDATE command. This case occurs with the command UPDATE record FROM keyval ... in which case a series of records could be updated by one command.

**Note:** Use of this option is not recommended because it violates the principle of the update cycle by committing records before all records in the update command have completed.

Default: N

### **Allow Reset READ FROM**

When this flag is set to Y then if the field used in a compiled FROM changes value then the READ FROM will reset its sequential position. This usually happens only when a new driving record has been read.

This option may also be set on a program basis using the RESET READ FROM clause (See IDENTIFICATION DIVISION ).

Programs must be recompiled when this flag is changed for the change to have any effect

Default: N

### **Pre-process Commands**

When this flag is set to Y all ad hoc commands will be preprocessed to look for any reserved word conversions.

Default: N

### **READ FROM Check Bypass**

When this flag is set to Y and the field used as the FROM specifier is a field in the record being read, TQL will return the first record that is greater than or equal to the key FROM specifier or PIB-EOF.

Default: N.

### **Failed Read Initializes**

When this flag is set to Y, the record area after a failed read contains initialized data (SPACES in alphanumeric and ZEROES in numeric).

When this is set to N, the contents of the record area after a failed read are left intact from any previous operations.

Default: N

### Use DOS Extensions

When this flag is set to Y, and an export to a file is performed. The file that the data is exported to may have a DOS extension. Otherwise if the flag is set to N, then the default extension for the output file is "PRN".

### Function Keys

The fields in this section allow the specification of the function key numbers corresponding to the advertised TQL function.

For example, to assign F8 to the "delete record" functionality of TQL, enter 8 in the field named "Delete:".

The default values are shown in the screen above.

## UF - Update File Definition

The UF command retrieves from TIP the current file information for the specified file. To see the information as TQL sees it use the L command.

If the user does not press the **TRANSMIT** key, (for example, presses the **CANCEL** key instead), the update file command is canceled without making any changes.

### Syntax:

**UF filename**

### Where:

**filename**

Name of the file that is to be updated.

## UP - Update Program Definition

The UP command updates an existing program definition.

When the user ends the editing session, TQLMON will automatically compile the program definition composed by the user. On successful compilation the source will be posted into the system. If any compile errors are encountered the source will not be posted.

### Syntax:

**UP progname**

### Where:

**progname**

The name of the TQL program that is being updated.

**Example:**

```
UP TESTPRO *> update program TESTPRO
```

## UPT, UT - Update Program or Record Template

The UPT command updates the source for a program template and UT updates the source for a record template.

When the user ends the editing session, TQLMON will post the template into the system.

**Syntax:**

```
UPT name
```

```
UT name
```

**Where:**

**name** The name of a template defined in the TQL system.  
The type of template listed is based on the command type.

**Example:**

```
UPT DEFAULT *> update the DEFAULT program template  
UT DEFAULT *> update the DEFAULT record template
```

## W - Write File/Record Definition to Source File

The W command writes the source for file or record definitions stored in the TQL system to a specified file. This function may be performed as part of a backup scheme or to facilitate transporting TQL definitions.

**Syntax:**

```
W file [ record ] [ unixfile ]
```

**Where:**

**file** The name of a file defined in the TQL system.

**record** The name of a record defined for the specified file.  
If this parameter is omitted the file definition for "file" is written.

**unixfile** UNIX output file. If this parameter is not supplied, a dialog box is displayed to prompt for the name using the value of the environment variable \$HOME as the initial destination.  
If this parameter is:

a directory name, a file named <record>.trd or <file>.tfd is written to that directory

a file name, the definition is written as that file name

**Example:**

Given that:

- /tmp is a directory
- outfile is a regular file

```
W PAYMAST PAYREC /tmp *> writes record
*> source to /tmp/PAYREC.trd
```

```
W PAYMAST /tmp *> writes file source to
*> /tmp/PAYMAST.tfd
```

```
W PAYMAST outfile *> writes file source to
*> outfile in the current directory.
```

## WP - Write Program Source to File

The WP command writes the source for program definitions stored in the TQL system to a specified file. This function may be performed as part of a backup scheme or to facilitate transporting TQL definitions.

**Syntax:**

```
WP progname unixfile
```

**Where:**

progname     The name of a program defined in the TQL system.

unixfile     UNIX output file. If this parameter is not supplied, a dialog box is displayed to prompt for the name using the value of the environment variable \$HOME as the initial destination. If this parameter is:

a directory name, a file named <progname>.tpd is written to that directory

a file name, the definition is written as that file name.

**Example:**

Given that:

- /tmp is a directory
- outfile is a regular file

```
WP PAYINQ /tmp *> write program source to
*> /tmp/PAYINQ.tpd
```

```
WP PAYINQ outfile *> write program to
*> outfile in the current directory
```

## WPT, WT - Write Program/Record Template to Source File

The WPT command writes the source for program template definitions stored in the TQL system to a specified file and the WT commands writes record template definitions. This function may be performed as part of a backup scheme or to facilitate transporting TQL definitions.

### Syntax:

```
WPT template unixfile
```

```
WT template unixfile
```

### Where:

#### template

The name of a template defined in the TQL system.

The type of the template written depends on the command used.

#### unixfile

UNIX output file. If this parameter is not supplied, a dialog box is displayed to prompt for the name using the value of the environment variable \$HOME as the initial destination.

If this parameter is:

- a directory name,  
a file named <template>.tpd or <template>.trd is written to that directory
- a file name,  
the definition is written as that file name.

### Example:

Given that:

- /tmp is a directory
- outfile is a regular file

```
WPT DEFAULT /tmp *> write program template
*> DEFAULT to /tmp/DEFAULT.tpd
```

```
WT DEFAULT /tmp *> write record template
*> DEFAULT to /tmp/DEFAULT.trd
```

```
WT DEFAULT outfile *> write record template
*> DEFAULT to outfile in the current
*> directory
```

## XF/XFC - Cross Reference Files

The XF command generates a report showing which TQL programs reference one or more specified files. The XFC command, omits the generated report but compiles all the programs that reference the file(s).

### Syntax:

```
XF [ file [ dest ] ]
```

```
XFC [ file ]
```

### Where:

- file** Name (or prefix) of TQL file to cross reference. Default: \* (all files).
- dest** TIPPRINT printer destination for output. Default: AUX0 (the terminal in full screen mode).

### Example:

```
XF TSPFILE *> show all programs using the file TSPFILE
XFC TSPFILE *> compile any programs using the
*> file TSPFILE
```

## XP - Cross Reference Programs

The XP command generates a report showing the files and records that are referenced by one or more TQL programs.

### Syntax:

```
XP [ prog ] [ dest ]
```

### Where:

- prog** Name (or prefix) of TQL program names to cross reference. Default: \* (all programs).
- dest** TIPPRINT printer destination for output. Default: AUX0 (the terminal in full screen mode).

### Example:

```
XP TQL* *> Show all files and records used
*> by programs beginning with TQL
```



## XR/XRC - Cross Reference Records

The XR command generates a report showing which TQL programs reference one or more specified record names. The XRC command omits the generated report but compiles all the programs that reference the record(s).

### Syntax:

```
XR [ file ] [ recd ] [ dest ]
```

```
XRC [ file ] [ recd ]
```

### Where:

- file** Name (or prefix) of TQL file to cross reference. Default: \* (all files).
- recd** Name (or prefix) of TQL record name to cross reference. Default: \* (all records).
- dest** TIPPRINT printer destination for output. Default: AUX0.

### Example:

```
XR PAYMAST PAYREC *> show all programs that  
*> use the record PAYMAT/PAYREC.
```

```
XRC TSPFILE TQLTSPR *> compile all programs  
*> that use the record TSPFILE/TQLTSPR.
```

# TQLADMIN - TQL System Administration

---

## TQLADMIN Features

TQL maintains information about the system by using a central control file as well as sub-directories under the \$TIPROOT/tql directory. The control file and files in the \$TIPROOT/tql directory should not be manipulated outside of the TQL environment.

This section describes initialization and verification of the TQL system and the target audience for this section is the TQL system administrator.

TQLADMIN is the administration program. It can be used to perform system initialization, verification, cleaning and rebuilding. Under most circumstances only the initialization functionality will be needed. If the TQL system is manipulated outside of the TQL environment then the system can become inconsistent. The verify, clean and rebuild functionality will put the system back into a consistent state.

These functions are described in greater detail in the following sections.

---

## Initializing the TQL Control File

The TQL system must be initialized before it is used for the first time. This initialization process catalogues the TQL control file to TIP, creates the file and writes a configuration record. The default templates records are also added.

### Syntax:

```
TQLADMIN INIT  
TQLINT
```

TQLINT is a synonym for TQLADMIN INIT.

On invocation of the initialization function there are four prompts to be answered.

### **Update Password**

This is asking for the password that will have to be entered when starting TQLMON if intending to make changes. If no password is entered then any user may start TQLMON in update mode.

See the TQLMON UC command for more information on the READ and WRITE passwords.

```
TQL$CTL already catalogued. Reset defaults
```

If the system has been initialized already then the control file catalogue entry will already exist. You may reset the entries in the SMFILE record to the system defaults.

This record is not usually modified once installed so resetting to the defaults will not destroy any information.

This prompt will not appear if this is the first time this program is being run.

**TQL\$CTL security record already exists. Reset defaults**

If the system has been initialized already then the control file security record will already exist. You may reset the entries in the SMSEC record to the system defaults.

This record is not usually modified once installed so resetting to the defaults will not destroy any information.

This prompt will not appear if this is the first time this program is being run.

**Control record already exists. Reset defaults**

If the system has been initialized already then the control record in the control file will already exist. You may reset the entries in this record to the system defaults.

This record contains information modifiable by users of [TQLMON](#) using the [UC](#) command. Items such as function key assignments, passwords and general flags are stored in this record. If the record is reset to the defaults then any customizations will be lost.

There are times when the control record will have to be reset. If the update password has been set but has been forgotten then the control record will have to be reset (using the new password supplied above). TQL also maintains the path of the system. If the control file is moved then the path will no longer be valid and TQL will fail to start. The control record will have to be rewritten.

This prompt will not appear if this is the first time this program is being run.

The following is the output from running the initialization function and answering yes to the prompts. No update password is being assigned at this point and the system has been initialized before.

```
TQLADMIN - TQL Control File Administration ( 99/03/25 2.3 R1 - 0000 )
UPDATE PASSWORD>
TQL$CTL already cataloged. Reset defaults? >Yes >No
TQL$CTL security record already exists. Reset defaults? >Yes >No
Control record already exists. Reset defaults? >Yes >No
```

If the answer to any query is "No", then you will be asked if you wish to continue. If you answered "No" the continue prompts the initialization process will be aborted. If you answered "No" because you didn't want

the data reset but do want to initialize the control file then answer "Yes" to the continue prompts.

The following is the output showing the results of answering yes and no to the continue prompts.

```
TQLADMIN - TQL Control File Administration (99/03/25 2.3 R1 - 0000)
UPDATE PASSWORD>
TQL$CTL already cataloged. Reset defaults? >Yes >No
(Yes selected)
TQL$CTL security record already exists. Reset defaults? >Yes >No
(No selected)
Continue? >Yes >No
(Yes selected)
Control record already exists. Reset defaults? >Yes >No
(No selected)
Continue? >Yes >No
(No selected)
Error initializing TQL.
```

---

## Checking the TQL Control File

The consistency of the control file may be verified at any time using the **CHECK** function of TQLADMIN. If the control file is incorrect for any reason this function will report the problem. This function simply checks the validity of the control file and makes no attempt to repair the file. To repair the system, see the **BUILD** and **CLEAN** functions described in the following sections.

The check involves verifying the contents of the control file against the TQL directory structure and then checking the directory structure against the control file. The majority of diagnostics are warnings because the system can recover from the problem. The only real error exists when there is a record in the control file but there is no associated source file.

### Syntax:

```
TQLADMIN[,options] CHECK [ dest ]
TQLCHECK[,options] [ ,dest ]
```

### Where:

#### options

The following options are accepted:

- O** Overwrite - Use of this option implies an affirmative answer to any prompt.
- Q** Quiet - Don't output any informational messages while checking.

**dest** Any valid TIPPRINT destination. The default is ROLL.

TQLCHECK is a synonym for TQLADMIN CHECK.

The following is the output from a sample run showing a number of diagnostics.

The count of records processed will always be greater than the sum of all the listed records because this count includes support records that are not shown. In the above example there is one support record and that happens to be the system control record. In most cases, the processed count will be one greater than the sum of the listed record types.

---

## Rebuilding the TQL Control File

The first control file repair function is the **BUILD** function. This scans the TQL directory structure for source files and adds the missing control file records.

On completion of the rebuild the TQL system should be recompiled to bring everything up to date. The last step of the rebuild will be to invoke the compiler to build the system.

Any existing invalid records in the control file are left intact. An invalid control file record is a record for which no valid source file exists. Missing ancillary files such as symbol table files do not invalidate the control file record because they can be recreated if the source file is present. See the following section for details on removing these invalid records.

There is an implied **CHECK** operation involved as part of the build.

It is highly recommended that **BUILD** is the first repair action taken when bringing the TQL system back into a consistent state. This reduces the risk of losing any required source files.

### Syntax:

```
TQLADMIN[,options] BUILD [ dest ]
TQLBUILD[,options] [ ,dest ]
```

### Where:

#### options

The following options are accepted:

- O** Overwrite - Use of this option implies an affirmative answer to any prompts. There are four possible replies to the prompts: "NO", "YES", "END" and "ALL". "END" is the same as "NO" but also stops any further building. "ALL" is the same as "YES" and indicates that all further queries are to be automatically answered "YES".
- Q** Quiet - Don't output any informational messages while building.

- R** Report - Display what actions would be performed by build. No changes are actually made. This allows for previewing the operations prior to rebuilding.

**dest** Any valid TIPPRINT destination. The default is ROLL.

TQLBUILD is a synonym for "TQLADMIN build".

The following is the output from a sample run requesting a report of actions to be taken.

```
TIP?>TQLBUILD,R
TQLADMIN - TQL Control File Administration ( 95/03/25 2.1 R1 - 0000 )
Add FILE record for tqlfiles/TSPFILE.tfd
Add RECORD record for tqlfiles/TSPFILE/TQLTSPR.trd
Add PROGRAM record for tqlprogs/TQLTSP.tpd
Warning: Program TIPSYS: Intermediate code missing.
Warning: FILE record missing for record TSPFILE/TQLTSPR2
Warning: Record TSPFILE/TQLTSPR2: Symbol table missing.
```

The following is the output from a sample **BUILD** run. In this sample all queries were answered YES.

```
TIP?>TQLBUILD
TQLADMIN - TQL Control File Administration ( 95/03/25 2.1 R1 - 0000 )
Add FILE record for tqlfiles/TSPFILE.tfd >NO >YES >END >ALL
Add RECORD record for tqlfiles/TSPFILE/TQLTSPR.trd >NO >YES >END >ALL
Add PROGRAM record for tqlprogs/TQLTSP.tpd >NO >YES >END >ALL
Warning: Program TIPSYS: Intermediate code missing.
Warning: Record TSPFILE/TQLTSPR: Symbol table missing.
Warning: Record TSPFILE/TQLTSPR2: Symbol table missing.
The following types were added:
Files : 1
Records : 1
Programs : 1
Invoke compiler to finish rebuilding? >YES >NO
Compiling records...
Compile successful.
Compiling programs...
Compile successful.
```

---

## Cleaning the TQL Control File

The second control file repair function is the **CLEAN** function. This functions will remove any invalid control file records and also remove any files in the TQL directory structure that should not be there.

An invalid control file record is a record for which no valid source file exists. Missing ancillary files such as symbol table files do not invalidate the control file record because they can be recreated if the source file is present.

When all the files and control file records have been removed the TQL system should be recompiled to bring everything up to date. The last step of the cleaning will invoke the compiler to recompile the system.

There is an implied **check** operation involved as part of the clean.

To reduce the risk of losing important source files it is recommended that the **BUILD** function be used prior to using **CLEAN**. See the previous section for more details on using **BUILD**.

**Syntax:**

```
TQLADMIN[,options] CLEAN [ dest ]
```

```
TQLCLEAN[,options] [ ,dest ]
```

**Where:**

**options**

The following options are accepted:

- O** Overwrite - Use of this option implies an affirmative answer to any prompts. There are four possible replies to the prompts: "NO", "YES", "END" and "ALL". "END" is the same as "NO" but also stops any further cleaning. "ALL" is the same as "YES" and indicates that all further queries are to be automatically answered "YES".
- Q** Quiet - Don't output any informational messages while cleaning.
- R** Report - Display what actions would be performed by clean. No changes are actually made. This allows for previewing the operations prior to cleaning.

**dest** Any valid TIPPRINT destination. The default is ROLL.

TQLCLEAN is a synonym for "TQLADMIN clean".

The following is the output of a sample **CLEAN** run answering all queries with "YES".

```
TIP??>tqlclean
TQLADMIN - TQL Control File Administration ( 95/03/25 2.1 R1 - 0000 )
Delete file tqlfiles/tqlmon >NO >YES >END >ALL
Delete file tqlfiles/tqlmon.debug >NO >YES >END >ALL
Delete file tqlprogs/EDITST/core >NO >YES >END >ALL
Delete file tqlprogs/TESTING/core >NO >YES >END >ALL
Delete file tqlprogs/TQLTSP/foo.lst >NO >YES >END >ALL
Delete file tqlprogs/TQLTSP/tqlmon.debug >NO >YES >END >ALL
```

---

## Clearing Edit Locks

If TQLMON has been terminated abruptly then an invalid edit lock will remain. This will prevent any further updates even though no one is

actively making changes any more. The **UNLOCK** function of TQLADMIN should be used to remove any invalid edit locks.

**Syntax:**

```
TQLADMIN[,options] UNLOCK [, dest ]
```

**Where:**

**options**

The following options are accepted:

- O** Overwrite - Use of this option implies an affirmative answer to any prompts. There are four possible replies to the prompts: "NO", "YES", "END" and "ALL". "END" is the same as "NO" but also stops any further cleaning. "ALL" is the same as "YES" and indicates that all further queries are to be automatically answered "YES".
- Q** Quiet - Don't output any informational messages while unlocking.
- R** Report - Display what actions would be performed by UNLOCK. No changes are actually made. This allows for previewing the operations prior to unlocking.

**dest** Any valid TIPPRINT destination. The default is ROLL.

The following is the output of a sample **UNLOCK** run answering all queries with "YES".

```
TIP??>TQLADMIN UNLOCK
TQLADMIN - TQL Control File Administration ( 95/03/25 2.1 R1 - 0000 )
Unlock PROGRAM TQLTSP >NO >YES >END >ALL
Unlock RECORD TSPFILE/TQLTSFR >NO >YES >END >ALL
```

---

## Setting Options in TQLADMIN

TQLADMIN options may be set using two methods. The first is to provide the options at invocation using the standard TIP method of specifying options. The second method is to use the **SET** command. This allows setting of options after TQLADMIN has started.

In addition, the **UNSET** command is used to clear an option.

Multiple options may be specified with the **SET** and **UNSET** commands but the options must be specified together without any separator characters.

**Syntax:**

```
SET [ option-list ]
```



**UNSET [ option-list ]**

**Where:**

option-list

The following options are accepted:

- O** Overwrite - Use of this option implies an affirmative answer to any prompts. There are four possible replies to the prompts: "NO", "YES", "END" and "ALL". "END" is the same as "NO" but also stops any further cleaning. "ALL" is the same as "YES" and indicates that all further queries are to be automatically answered "YES".
- Q** Quiet - Don't output any informational messages.
- R** Report - Display what actions would be performed by any requested operation.

When specifying multiple options all the options being SET or UNSET must be specified together without separator characters (i.e. SET RQ)

If no options are specified for the set command the list of currently set options will be displayed.

**Example:**

```
TQLADMIN(1)>SET R *> set Report option

TQLADMIN(1)>BUILD *> invoke BUILD to find out what will
*> be added
*> BUILD output deleted for clarity

TQLADMIN(1)>UNSET R *> remove Report option

TQLADMIN(1)>BUILD *> invoke BUILD
*> BUILD output deleted for clarity

TQLADMIN(1)>SET OQ *> set Overwrite and Quiet options

TQLADMIN(1)>CLEAN *> invoke CLEAN
```

---

## Converting Saved Command Files

Since release 2.2 of TIP/ix, TQL can track some extra information for a saved command. TQL will now track the number of times the command was recalled, when it was recalled and by whom.

To enable this option the old style saved command file should be updated and converted to the new format. This is done with the CONVTQL script located in \$TIPROOT/scripts.

TQL can continue to work with the previous format. It is not necessary to convert to the new format. However, once converted you will not be able to convert back.

To convert to the new format you must be at the UNIX prompt. You will change directories to ***\$TIPROOT/src/tql*** and enter:

```
$> tipix -s CONVTQL
```

This will start TIP and the conversion program. Once running, you may be prompted with questions which you should answer YES to.

Once the script is finished running, TQL commands that were previously saved in the file TQLSVE will have been transferred to TQLSVE2 and have the new features mentioned above.

The saved TQL commands can still be viewed via the TQLSVE program, which will now reference the TQLSVE2 file.

---

## Exiting TQLADMIN

TQLADMIN will enter command mode if the transaction invoked was TQLADMIN and not one of the short forms such as TQLCLEAN. Use any one the following commands to exit from TQLADMIN.

**Syntax:**

```
E  
END  
Q  
QUIT  
FIN
```

## TQLSVE - TQL Saved File Maintenance

This section describes the saved command maintenance TQL program TQLSVE. This TQL program is provided for your convenience by IngleNet Business Solutions and allows for TQL program independent manipulation of the saved command file. This program is generally used by the TQL system administrator. The end user should not need to use this program.

See the TQLRUN RECALL and EXECUTE commands for more details on using saved commands in a TQL program.

The saved command file is an ISAM file with a sixteen byte key being comprised of the program name and command identifier. It is logically defined to TIP as TQLSVE and the record description is defined by the record SVEREC.

The saved command file is automatically available to the TQL system and does not depend on installing this TQL program.

---

### Installing TQLSVE

Before the program TQLSVE can be used it must be installed into the TQL system. This step is usually done by the system administrator at TIP installation time.

The following commands can be used to install the TQLSVE program if it has not already been installed.(assumes you at the TIP prompt):

- ▶TQLMON
- ▶AF TQLSVE
- ▶C SVEREC.trd
- ▶CP TQLSVE.tpd

The program is now installed and available for use.

See the TQLMON section for more details on the [AF](#), [C](#) and [CP](#) commands used above.

---

### Running TQLSVE

Once the program TQLSVE has been installed in the TQL system it may be used to view or print the saved commands. See the TQLRUN section regarding for detailed information on running TQL programs in general and the runtime command set.

To run TQLSVE enter the following at the TIP prompt:

**►OPEN TQLSVE**

TQLSVE has one defined display and one report. The display CMD shows a saved command and this saved command may be modified using this display. The report PRTCMD will print all records accordingly to PRNTR.

Running the display CMD will produce output similar to the following if there are any saved commands present.

## TQL Example Programs

### Inventory/Order Example

This section illustrates many of the features of TQL. The example shows the file and record definitions for a simple inventory file and associated order file. The programs illustrated provide the capability to maintain the inventory file (INV) and the order file (ORD) and to enter orders, change orders, display orders, print orders etc., while keeping track of inventory.

#### Inventory File

The inventory file has a logical file name of "INV" in the TIP catalogue and has the following characteristics:

```
FILE INV,MIRAM BLKSIZE=500
RECSIZE=50
KEY1=(4,0,NDUP,NCHG)
KEY2=(16,4,DUP,CHG)
KEY3=(2,20,DUP,CHG)
ACCESS=EXCR.
RECORD INVREC
01 INVREC.
05 INV-PART PICTURE 9(4).
05 INV-DESC PICTURE X(16).
05 INV-LOC PICTURE 99.
05 INV-QOH PICTURE 9(5).
05 INV-PRICE PICTURE 9(5)V99.
ALLOW CHANGE ALL.
ALLOW ADD.
ALLOW DELETE.
```

The primary key of a MIRAM file must not allow duplicates or changes (TIP restriction).

The primary key is the inventory part number.

This example system also makes use of an order file (logical file name "ORD") which has the following characteristics:

```
FILE ORD,MIRAM BLKSIZE=1000
RECSIZE=100
KEY1=(16,0,NDUP,NCHG)
ACCESS=EXCR.
```

#### Order File

The order file contains two types of records:

A header record (one per order).

```

RECORD ORDHDR
01 ORDHDR.
05 HDR-KEY.
10 HDR-ORD.
15 HDR-CUST PICTURE X(8) .
15 HDR-NUM PICTURE 9(4) .
10 HDR-LINE PICTURE 9(4) .
05 HDR-PO-NUM PICTURE X(8) .
05 HDR-LAST-LINE PICTURE 9(4) .
ID IS HDR-LINE = 0.
ALLOW CHANGE ALL.
ALLOW ADD.
ALLOW DELETE.

```

A header record is distinguished by the field HDR-LINE equal to zero.

2. A detail record (one or more per order - representing items ordered):

```

RECORD ORDDTL
01 ORDDTL.
05 ORD-KEY.
10 ORD-CUST PICTURE X(8) .
10 ORD-NUM PICTURE 9(4) .
10 ORD-LINE PICTURE 9(4) .
05 ORD-PART PICTURE 9(4) .
05 ORD-QTY PICTURE 9(4) .
ID IS ORD-LINE > 0.
MUST ADD ORD-QTY.
ALLOW CHANGE ALL.
ALLOW ADD.
ALLOW DELETE.

```

A detail order record (representing on item ordered) is distinguished by the field ORD-LINE greater than 0. The field is incremented by one for each item in the order (items 1 through last item).

The following TQL program was written to provide maintenance capabilities for the inventory file. Two predefined displays are defined by the program:

"PART " display (all fields) in a single inventory (part) record

"PARTS" display (all fields) in five inventory records.

The screen formats "TF\$TQDM1" and "TF\$TQDM2" are shown following the program source.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. INV 'INVENTORY UPDATE' .

```

```

DATA DIVISION.
FILE INV.
RECORD INVREC.
DISPLAY DIVISION.
PART: READ INVREC
INVREC
USING TF$TQDM1.
PARTS: 5 { READ INVREC
INVREC }
USING TF$TQDM2.
    
```

The main processing program (shown below) is used to enter new orders, perform maintenance operations on existing orders and (in all cases) adjust the quantity on hand in the inventory file according to the number of items ordered or returned.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. ORD.
DATA DIVISION.
FILE ORD.
RECORD ORDHDR.
RECORD ORDDTL.
FILE INV.
RECORD INVREC.
FILE TSPFILE.
RECORD TQLTSPR.

WORKING-STORAGE SECTION.
01 WORK-AREA.
05 TOT-PRICE PIC 9(5)V99.
05 PREV-QTY PIC 9(4).
DECLARATIVES SECTION.
ON READ OF ORDHDR
READ TQLTSPR VIA HDR-CUST.
ON READ OF ORDDTL
MOVE ORD-QTY TO PREV-QTY
READ INVREC VIA ORD-PART.
ON WRITE OF ORDHDR
    READ TQLTSPR VIA HDR-CUST
ON ERROR 'INVALID CUSTOMER #'.
ON WRITE OF ORDDTL
    READ INVREC VIA ORD-PART
ON ERROR 'BAD PART NUM'
    READ TQLTSPR VIA ORD-CUST
ON ERROR 'INVALID CUST #'
MOVE ORD-CUST TO HDR-CUST
MOVE ORD-NUM TO HDR-NUM
MOVE 0 TO HDR-LINE
    
```

```

READ ORDHDR VIA HDR-KEY
ON ERROR 'MISSING HEADER RECORD'
IF INV-QOH < ORD-QTY
ERROR 'NOT ENOUGH GOODS'
COMPUTE INV-QOH = INV-QOH + PREV-QTY - ORD-
QTY
MOVE ORD-LINE TO HDR-LAST-LINE.
DISPLAY DIVISION.
NEWORDER: READ ORDHDR
HDR-CUST HDR-NUM HDR-PO-NUM
USING TF$TQDM3 ON ENTER ORDER.
ORDER: MOVE HDR-CUST TO ORD-CUST
MOVE HDR-NUM TO ORD-NUM
MOVE HDR-LAST-LINE + 1 TO ORD-LINE
READ ORDDTL
  ORDDTL
  USING TF$TQDM3.
ORDDISP: READ ORDHDR
HDR-CUST HDR-NUM HDR-PO-NUM CM-COMPANY NL$
8 { READ ORDDTL FROM HDR-ORD
ORD-LINE ORD-PART INV-DESC ORD-QTY INV-
PRICE
COMPUTE TOT-PRICE = INV-PRICE * ORD-QTY
TOT-PRICE NL$ }
USING TF$TQDM5.

```

---

## ORD Program Description

- Whenever an order detail record is read, the number of items ordered (ORD-QTY) is saved in working-storage field "PREV-QTY". This is done so that the quantity on hand in inventory can be recalculated if the detail item is updated (or deleted).
- Whenever an order detail record is written this coding validates the part number and the customer number according to the data in other files.
- It also verifies that there is an existing header record for this detail record.
- If the quantity-on-hand in the INV file (INV-QOH) is not sufficient an error message is produced ("NOT ENOUGH GOODS")
- Finally, the inventory quantity on hand is recalculated and the inventory file is updated too.
- The display "NEWORDER" is used to enter a new order. The "ON ENTER" clause specifies that when the user has entered the data in screen "TF\$TQDM3" he/she is to be taken (in data entry mode) to pre-defined display "ORDER".



- The display "ORDER" therefore, is chained to the entry of a new order.
- The display "ORDER" is used as described above (as a secondary activity of order entry). It may also be used directly to perform maintenance activities on order detail records.
- The display "ORDDISP" displays the header information for an order and displays (on the same screen) up to eight order detail records.

---

## INVOICE Program

The following TQL program was written to generate invoices from the orders in the order file. A number of sample invoices produced by this program (using test data) are shown following the program source.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. INVOICE.
DATA DIVISION.
FILE ORD.
RECORD ORDHDR.
RECORD ORDDTL.
FILE INV.
RECORD INVREC.
FILE TSPFILE.
RECORD TQLTSPR.

WORKING-STORAGE SECTION.
01 WORK-AREA.
05 TOT-PRICE PIC 9(5)V99.
05 GRAND PIC 9(6)V99.
05 TAX PIC 9(6)V99.
05 FINAL PIC 9(6)V99.
05 SUM-TAX PIC 9(7)V99.
05 SUM-DUE PIC 9(7)V99.
05 SUM-GOODS PIC 9(7)V99.
DECLARATIVES SECTION.
ON READ OF ORDHDR
READ TQLTSPR VIA HDR-CUST.
ON READ OF ORDDTL
READ INVREC VIA ORD-PART.
REPORT DIVISION.
INVOICE: READ ORDHDR HOME$
TAB$(10) 'SAMPLE ORDER INVOICE'
SKIP$(4) YY$/'MON$'/'DD$ NL$ NL$
' CUST # ORD# P.O.# COMPANY NAME' NL$
HDR-CUST ' ' HDR-NUM ' ' HDR-PO-NUM ' ' CM-
COMPANY

```

```

NL$ NL$
' LINE PART# DESCRIPTION'
TAB$(34) 'QUANTITY PRICE TOTAL'
NL$ 50 {READ ORDDTL FROM HDR-ORD
ORD-LINE ' '
ORD-PART ' '
INV-DESC ' '
ORD-QTY ' '
INV-PRICE ' '
COMPUTE TOT-PRICE = INV-PRICE * ORD-QTY
TOT-PRICE
ADD TOT-PRICE TO GRAND NL$} NL$
COMPUTE TAX = GRAND * 0.07
COMPUTE FINAL = GRAND + TAX
TAB$(41) 'TOTAL PRICE ' GRAND NL$
TAB$(41) ' SALES TAX ' TAX NL$
TAB$(41) ' AMOUNT DUE ' FINAL NL$
ADD TAX TO SUM-TAX
ADD FINAL TO SUM-DUE
ADD GRAND TO SUM-GOODS
ON PRNTR
AT END HOME$ NL$ NL$
'TOTAL VALUE OF GOODS SOLD' SUM-GOODS NL$
'TOTAL TAX DUE GOVERNMENT' SUM-TAX NL$
'TOTAL AMOUNT TO COLLECT' SUM-DUE NL$.

```

The following report was produced by the program INVOICE.

```

SAMPLE ORDER INVOICE 83/06/01
CUST # ORD# P.O.# COMPANY NAME
COA00000 1 BILL CITY OF ARVADA
LINE PART# DESCRIPTION QUANTITY PRICE TOTAL
1 3 RED SHIRT 4 22.50 90.00
2 1 WHITE SHIRT 1 11.95 11.95
3 3 RED SHIRT 4 22.50 90.00
4 7 THIN TIE 3 2.00 6.00
5 11 NEHRU JACKETS 3 1.95 5.85
TOTAL PRICE 203.80
SALES TAX 14.27
AMOUNT DUE 218.07

```

===== new page =====

```

SAMPLE ORDER INVOICE 83/06/01
CUST # ORD# P.O.# COMPANY NAME
GLO00000 1 DAVID GENERAL LAND OFFICE
LINE PART# DESCRIPTION QUANTITY PRICE TOTAL
1 3 RED SHIRT 5 22.50 112.50
2 5 WIDE TIE 8 6.50 52.00

```

3 7 THIN TIE 3 2.00 6.00  
TOTAL PRICE 374.30  
SALES TAX 26.20  
AMOUNT DUE 400.50

===== new page =====

SAMPLE ORDER INVOICE 83/06/01  
CUST # ORD# P.O.# COMPANY NAME  
GLO00000 56 XYZ GENERAL LAND OFFICE  
LINE PART# DESCRIPTION QUANTITY PRICE TOTAL  
1 5 WIDE TIE 4 6.50 26.00  
2 1 WHITE SHIRT 7 11.95 83.65  
TOTAL PRICE 483.95  
SALES TAX 33.88  
AMOUNT DUE 517.83

===== new page =====

TOTAL VALUE OF GOODS SOLD 1062.05  
TOTAL TAX DUE GOVERNMENT 74.35  
TOTAL AMOUNT TO COLLECT 1136.40

# ANSI COCOL-85 Specifications

The following sections were reproduced for your convenience from:

**American National Standard for Information Standards -  
Programming Language - COBOL  
ANS Institute, New York, 1985.**

---

## Qualification

Every user-defined name explicitly referenced in a COBOL source program must be uniquely referenced because either:

No other name has the identical spelling and hyphenation.

It is unique within the context of a REDEFINES clause.

The name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher level names in the hierarchy.

These higher level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of redefinition. All available qualifiers need not be specified so long as uniqueness is established. Reserved words naming the special registers require qualification to provide uniqueness of reference whenever a source program would result in more than one occurrence of any of these special registers. A paragraph-name or section-name appearing in a program may not be referenced from any other program.

---

## Reference Modification

### Function

Reference modification defines a data item by specifying a leftmost character and length for the data item.

### General Format

```
data-name-1 (leftmost-character-position: [length])
```

## Syntax Rules

Data-name-1 must reference a data item whose usage is DISPLAY.

Leftmost-character-position and length must be arithmetic expressions.

Unless otherwise specified, reference modification is allowed anywhere an identifier referencing a data item of the class alphanumeric is permitted.

data-name-2 may be qualified or subscripted.

## General Rules

Each character of a data item referenced by data-name-1 is assigned an ordinal number incrementing by one from the leftmost position to the rightmost position. The leftmost position is assigned the ordinal number one. If the data description entry for data-name-1 contains a SIGN IS SEPARATE clause, the sign position is assigned an ordinal number within that data item.

If the data item referenced by data-name-1 is described as numeric, numeric edited, alphabetic, or alphanumeric edited, it is operated upon for purposes of reference modification as if it were redefined as an alphanumeric data item of the same size as the data item referenced by data-name-1.

Reference modification for an operand is evaluated as follows:

- a) If subscripting is specified for the operand, the reference modification is evaluated immediately after evaluation of the subscripts.
- b) If the subscripting is not specified for the operand, the reference modification is evaluated at the time subscripting would be evaluated if subscripts had been specified.

Reference modification creates a unique data item which is a subset of the data item referenced by data-name-1. This unique data item is defined as follows:

- a) The evaluation of leftmost-character-position specifies the ordinal position of the leftmost character of the unique data item in relation to the leftmost character of the data item referenced by data-name-1. Evaluation of leftmost-character -position must result in a positive non-zero integer less than or equal to the number of characters in the data item referenced by data-name-1.
- b) The evaluation of length specifies the size of the data item to be used in the operation. The evaluation of length must result in a positive non-zero integer. The sum of leftmost-character-position and length minus the value one must be less than or equal to the number of characters in the data item referenced by data-name-1. If length is not specified, the unique

data item extends from and includes the character identified by leftmost-character-position up to and including the rightmost character of the data item referenced by data-name-1.

The unique data item is considered an elementary data item without the JUSTIFIED clause. It has the same class and category as that defined for the data item referenced by data-name-1 except that the categories numeric, numeric edited, and alphanumeric edited are considered class and category alphanumeric.

## Identifier

An identifier is a term used to reflect a data-name that, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or reference modifiers necessary for uniqueness of reference.

# Index

Contents ..... i