

# PowerCL<sup>TM</sup>

## Control Language for Files (CLF) Programmer's Guide for RPG Developers

Bruce Vining Services, LLC.

*Integrated Solutions for the  
i user community*

2253 5<sup>th</sup> Ave NE  
Rochester, MN 55906  
Phone 507/206-4178  
[www.brucevining.com](http://www.brucevining.com)

Copyright 2009 Bruce Vining Services, LLC

CLF02-01-00

## COPYRIGHT

© Copyright 2009 Bruce Vining Services, LLC. All rights reserved.

Both this book and the software described by this book are protected by copyright. You may not copy or reproduce this book in any form without prior written permission from Bruce Vining Services, LLC. The software associated with this product is governed by a license agreement. This software is yours to use only as long as you adhere to the terms of the license agreement.

The following items in PowerCL Control Language for Files (CLF) are protected by copyright law:

- The CLF Programmer's Guide for CL Developers
- The CLF Programmer's Guide for RPG Developers
- The CLF Installation Reference
- The CLF Run-Time Generation Tools Guide
- The product brochure
- All text and titles on the software's entry and display panels, including the look and feel of the interaction of the panels along with the supporting menus, pop-up windows, and function key descriptions and layout.

PowerCL is a trademark of Bruce Vining Services, LLC.

Any individuals or corporations who violate these copyrights and trademarks will be prosecuted under both criminal and civil laws and any resulting products will be required to be withdrawn from the marketplace.

The following are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM i5/OS  
System i

This product contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by and actual business enterprise is entirely coincidental.

This product contains sample application programs in source language that illustrate programming techniques. You may copy, modify, and distribute these sample programs in any form without payment to Bruce Vining Services, for the purposes of developing, using, marketing, or distributing application programs. These examples have not been thoroughly tested under all conditions. Bruce Vining Services, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

## Contents

1. WELCOME TO CONTROL LANGUAGE FOR FILES .....	7
Summary of Features .....	7
2. ABOUT THIS MANUAL .....	10
3. PRODUCT PACKAGING, INSTALLATION, AND MAINTENANCE .....	12
Product Packaging.....	12
CLF Base Product Option.....	12
CLF Option 1: The Precompiler .....	13
CLF Option 2: Run-Time Generation Tools.....	13
CLF Option 3: Sample Programs.....	13
Installation, Upgrading, and Maintaining .....	15
CLF Product Support .....	15
4. CREATING AND RUNNING A CLF APPLICATION .....	16
Getting Started.....	19
Randomly Read a Record by Key .....	22
Writing a Record to a Database File .....	26
Updating a Database Record .....	30
Deleting One or More Database Records .....	34
Using a Display File.....	39
Using a Printer File.....	45
Why Develop Using the CLF Precompiler?.....	50
Enhanced Productivity .....	50
CLF Commands Not Available Outside the Precompiler Environment .....	51
Compiling a CLF Program .....	54
Using the CRTBNDCLF Command.....	54
Using the CRTCLFMOD Command .....	55
Using the CRTCLFPGM Command.....	56
Conventional Thinking Concerning File Usage.....	58
5. FILES, RECORD FORMATS, AND FIELDS.....	63

Using Externally Described Files .....	63
Using the FILE and DCLFCLF command .....	63
Using the GENFFDCLF command .....	67
Using Program Described Files .....	70
Using Externally and Program Described Files in the Same Program.....	71
Using CLF and CL Files in the Same Program .....	72
Field Initialization Considerations.....	75
6. INDICATORS .....	78
Using the INDS Command.....	79
Using the GENINDCLF Command.....	80
Using the GENINDCLF Command and the Precompiler for the Same Program .....	80
7. DETECTING FILE ERRORS AND CONDITIONS .....	82
Messages.....	82
Obtaining additional information from CLF messages .....	84
Program variables .....	88
Having CLF declare your program variables for you .....	89
Messages and Program Variables .....	89
8. USING DATABASE FILES .....	91
Access Paths.....	94
Key Values .....	98
Key Relation (KEYREL) Examples .....	100
Read examples .....	100
Positioning examples .....	114
Example using Database Read, Delete, and Update with the CLF Precompiler .....	124
Using OPNQRYF.....	128
OPNQRYF with Record Selection .....	128
OPNQRYF with Join, Record Grouping, and Mapped Fields .....	130
Commitment Control .....	133
Trigger Programs.....	134
Differences From Conventional CL File Usage .....	135
CL Variable Types .....	135

Opening a Database File .....	136
End-of-File Processing.....	136
Closing a Database File .....	136
Database File Capability Language Comparison .....	137
9. USING DISPLAY FILES.....	141
Working with Subfiles .....	143
Work with Print Control.....	144
Using the Precompiler and CL File Support in the Same Application .....	157
Display File Capability Language Comparison.....	159
10. USING PRINTER FILES .....	160
External Printer Files .....	163
Example using External Print File and the CLF Precompiler .....	163
Program-Described Printer Files.....	167
*FCFC Processing .....	168
*Machine Processing .....	174
Printer File Capability Language Comparison .....	181
APPENDIX A. ILE RPG AND CLF.....	182
RPG and CLF File Support and Declaration .....	182
RPG and CLF File Operations Mapping .....	184
RPG and CLF File Operation Code Extender Mappings.....	185
RPG and CLF Miscellaneous Mappings .....	186
APPENDIX B. VC2EMP AND VC2DPT SAMPLE FILES .....	187
VC2EMP description and logical files.....	188
VC2DPT description and logical files.....	191
Loading Records into the Sample Files .....	193
Instructions for the RPG_LOAD program.....	193
Instructions for loading the VC2DPTNUL sample file.....	193
Instructions for loading the VC2DPTVAR sample file.....	194
APPENDIX C. COMPILE ERRORS USING THE PRECOMPILER.....	195
APPENDIX D. CLF COMMANDS.....	196
RPG-like syntax style .....	196



# 1. Welcome to Control Language for Files

The IBM i Control Language (CL) is a very powerful and productive language that virtually all programmers and operators of the system are familiar with. It allows you to run your system, write application programs to automate system operations, and write application programs work with the products you may have installed on your system. CL however does have limitations in terms of its ability to work with the business databases that run your company and its ability to provide information to users in the form of interactive applications and printed reports.

The PowerCL Control Language for Files (CLF) product addresses these limitations and does so in a manner that maximizes the productivity of your operators and programmers. CLF supports the use of System i files directly from CL applications. With a rich set of CL commands CLF allows your CL developers to directly work with file types such as database, display, and printer. System operators and application developers can experience significant productivity enhancements as they no longer need to work with multiple languages in order to use the flexibility of CL and the database support of high level languages such as RPG, COBOL, and C. CLF provides a superset of the file support found in these other high level languages, and this superset is accessible entirely through CLF provided command interfaces. CLF provides a common, consistent interface to your System i resources.

## SUMMARY OF FEATURES

- Supports physical files, logical files, DDM files, Open query files, and SQL views
  - Externally described data in addition to program described
  - Keyed access and access by relative record number
  - Read, Write, Update, and Delete
  - Dynamic changes in sequential read direction
  - Commit and Rollback support
  - Support for null fields, variable-length fields, extended data types, reuse of deleted records
  - Logical files can be single format, multi-format, or join
- Supports display files

- Externally described data in addition to program described
- Subfiles
- Read, Write, and Write/Read support
- Multiple device support
- Supports printer files
  - Externally described data in addition to program described
  - Override of spool file name, user data
- Two command interfaces provided
  - Traditional CL syntax such as:
    - OPNFCLF to Open a File using CLF
    - POSDBFCLF to Position a File using CLF
  - Free-form RPG-like syntax such as:
    - OPEN to Open a File using CLF
    - SETLL to Position a File using CLF
  - Can intermix the two syntax styles to maximize development productivity
- Extensive on-line help for commands and menus
- CLF file commands can co-exist with traditional CL file commands
  - Add CLF function to existing CL applications with minimal effort
  - Greater productivity by not having to use the OPNID of traditional CL DCLF support when using more than one file in the CL application program
- Elimination of need to write RPG/COBOL/C programs to meet input/output needs of CL application
  - Less programs to develop/test/maintain

- Easier application development by staying in one application language environment. If 95% of the function is best served by traditional CL why introduce RPG, COBOL, or C for the remaining 5%?
- Supports both Integrated Language Environment (ILE) and Original Program Model (OPM) CL program environments
- Sample databases and complete application examples provided
- Capability of writing non-trivial input/output applications entirely in CL
- National language enabled and will operate correctly with any System i supported national language version (NLV)
- Supports up to 1000 concurrently open files per activation group, over 32000 concurrently open files per job

## 2. About This Manual

The CLF Programmer's Guide for the RPG Developer provides information that shows how to use the Control Language for Files (CLF) precompiler, run-time development tools, and run-time. This manual explains how to use CLF assuming that you are using the precompiler and RPG-like syntax commands (that is, commands such as SETLL and CHAIN rather than conventional CL syntax commands such as POSDBFCLF and READRCDCLF). CL syntax commands will be used when there is no equivalent RPG operation code for a particular CLF capability. When using CLF you can intermix CLF RPG-like commands and CLF CL-type commands in any manner that meets your application development needs. In some cases you will find that using a conventional CL syntax command can save you time and effort.

This manual does not provide Reference Manual level of information. The online help text associated with each CLF command provides that level of detail. For access to reference level information when you do not have access to a Power i system you will find the command documentation is also available at [www.brucevining.com](http://www.brucevining.com).

This guide is for programmers who are already familiar with Control Language (CL) and i file support, and want to learn how to use i files from a CL based application. This manual does not provide introductory information related to how to enter CL source statements, run CL programs, or write CL programs. For this type of knowledge Bruce Vining Services can provide education to your company. Alternatively you can refer to the IBM Information Center, IBM educational offerings, and/or several introductory books that have been written on CL.

Your feedback on CLF and this manual is important in helping to provide the most accurate and high-quality information. Bruce Vining Services welcomes any comments about this manual or other Bruce Vining Services documentation.

If you prefer to send comments by mail, use the following address:

Bruce Vining Services  
Attn: Information Development  
2253 5<sup>th</sup> Ave NE  
Rochester, MN 55906

If you prefer to send comments electronically, please go to our web site at [www.brucevining.com](http://www.brucevining.com) and select 'Contact Us'.

Be sure to include the name of the book, the version of the book (found on the front page, lower left corner) and the page number or topic that your comment applies to.

This manual is available online at [www.brucevining.com](http://www.brucevining.com). When working with a hardcopy version of this manual you will want to verify that your copy is the most recent.

# 3. Product Packaging, Installation, and Maintenance

## PRODUCT PACKAGING

CLF is a member of the PowerCL family of products. The CLF product, 1BVSCLF, is installed using the Restore Licensed Program (RSTLICPGM) command. PTFs are applied using the Apply Program Temporary Fix (APYPTF) command. The CLF product has four options (or features) defined:

- The base run-time support
- A precompiler which provides for maximum productivity when developing CL database, interactive, and/or report generating applications
- A set of commands to generate file, field, and indicator definitions when developing CL database, interactive and/or report generating applications without the use of the precompiler
- Sample source code for CLF programs, database files, display files, and printer files

You can download a fully functional trial of CLF from the web site at [brucevining.com](http://brucevining.com).

### ***CLF Base Product Option***

The CLF base product provides run-time environment support for CLF applications. CLF applications can be developed on one system (using the optional precompiler support, the optional generation commands, or the base run-time support) and then distributed to other systems that have only the base run-time product installed. The base product option provides all necessary run-time support for *any* CLF application.

In addition to the run-time commands to work with files the base product also provides all of the CLF precompiler commands with the exception of the actual create commands - - CRTBNDCLF, CRTCLFPGM, CRTCLFMOD, CLFI, and CLFO. The remaining precompiler commands are packaged with the base product so that you can prompt and access the online help provided with these commands if you are performing problem determination on a production system that does not have the precompiler installed. A list of the commands provided with this option can be found in the CLF Installation Reference manual.

Though there is nothing to prevent a developer from writing an application using only the base run-time support, programmer productivity will not be high relative to using either a precompiler or the generation tools. The intent of providing the base product as a separate, no charge option is to facilitate the development of applications using the precompiler, or the generation tools, and then distributing those applications to production environments where no additional fee-based software is required. It is not intended that developers create applications using only the base run-time product.

### ***CLF Option 1: The Precompiler***

Option 1 of the CLF product provides precompilers for both OPM and ILE CL application developers. The precompilers provide a rich development environment for the building of CLF applications -- an environment that can significantly improve the productivity of CL developers when working with files on the System i. The precompilers support both a RPG-like command syntax and a conventional CL command syntax. The CLF precompilers greatly simplify the development effort involved when working in environments such as common field names that exist in more than one file (in particular common field names across database files, display files, and/or printer files), multi-format files, database files using fields defined as null capable or variable-length, etc. A list of the commands provided with this option can be found in the CLF Installation Reference manual.

### ***CLF Option 2: Run-Time Generation Tools***

Option 2 of the CLF product provides support for externally described files when the precompiler (option 1) is not available to you. This option provides a level of development function that is sufficient for the creation of simple CLF applications. An example of a simple application is one where only one file is needed within the CL program. Multi-file applications can be written using only the base run-time support and the Run-time Generation Tools, but programmer productivity will not be as high when compared to using the precompiler support. A list of the commands provided with this option can be found in the CLF Installation Reference manual. When ordering a license key for option 1 of CLF you automatically also receive a license key for option 2.

### ***CLF Option 3: Sample Programs***

Option 3 of the CLF product provides a large number of sample programs in the CLF product library VC2CLF. The source files are VC2CLSRC, QDDSSRC, and QCMDSRC.

VC2CLSRC contains the source for the sample programs used in this manual and the example programs provided in the online help available with all CLF commands.

- Source members with a name starting with 'DEV\_' are example programs that require the use of the precompiler of option 1 in order to be created. Once compiled, the DEV\_ programs require only the run-time base support of CLF in order to run.
- Source members with a name starting with 'RPG\_' are example programs that require the use of the precompiler of option 1 and demonstrate the use of the free-form RPG-like command interface. Once compiled, the RPG\_ programs require only the run-time base support of CLF in order to run.
- Source members with a name starting with 'RNT\_' are example programs that utilize the run-time generation tools of option 2. These examples can be compiled using the i commands Create Bound CL Program (CRTBNDCL), Create CL Module (CRTCLMOD), or Create CL Program (CRTCLPGM). Once compiled, the 'RNT\_' programs require only the run-time base support of CLF in order to run.
- Source members with a name starting with 'BAS\_' are example programs that are written using only the base-runtime support. Once compiled, the BAS\_ programs require only the run-time base support of CLF in order to run.

The QDDSSRC source file contains:

- DDS for the sample databases that are provided with the CLF product
- DDS for display files that are used by some of the CLF sample programs
- DDS for printer files that are used by some of the CLF sample programs

The QCMDSRC source file contains the source for commands that are related to some of the example programs found in this manual.

A list of the objects provided with this option can be found in the CLF Installation Reference manual.

## **INSTALLATION, UPGRADING, AND MAINTAINING**

To install one or more options of the CLF product, upgrade to the latest release of the CLF product, or apply PTFs to the CLF product refer to the CLF Installation Reference manual.

### **CLF PRODUCT SUPPORT**

The level of product support available to you depends on the options that you have installed on your system.

When you install the precompiler (option 1 of CLF) license you receive 6 months of software support. This support includes telephone and electronic access to highly-trained technical support specialists to provide fast, accurate problem resolution to help keep your IT staff productive. Included is software defect support for reporting suspected problems and electronic access to PTFs correcting resolved problems. These problems can be related to any option of the CLF product. After the initial 6 months a subscription plan is available to continue receiving telephone access, electronic access, and problem reporting support. Without a subscription you will continue to have electronic access to PTFs correcting resolved problems.

When you install the run-time generation tools (option 2 of CLF) license (without the precompiler) you receive 3 months of software support. This support includes electronic access to highly-trained technical support specialist to provide fast, accurate problem resolution to help keep your IT staff productive. Included is software defect support for reporting suspected problems and electronic access to PTFs correcting resolved problems. These problems can be related to the generation tools or CLF base run-time. After the initial 3 months a subscription plan is available to continue receiving electronic access and problem reporting support. Without a subscription you will continue to have electronic access to PTFs correcting resolved problems.

When you install the run-time base support, without either the precompiler or the run-time generation tools, you receive electronic access to PTFs correcting resolved problems.

Additional support options, including on-site education for your staff, is available through extended support offerings.

## 4. Creating and Running a CLF Application

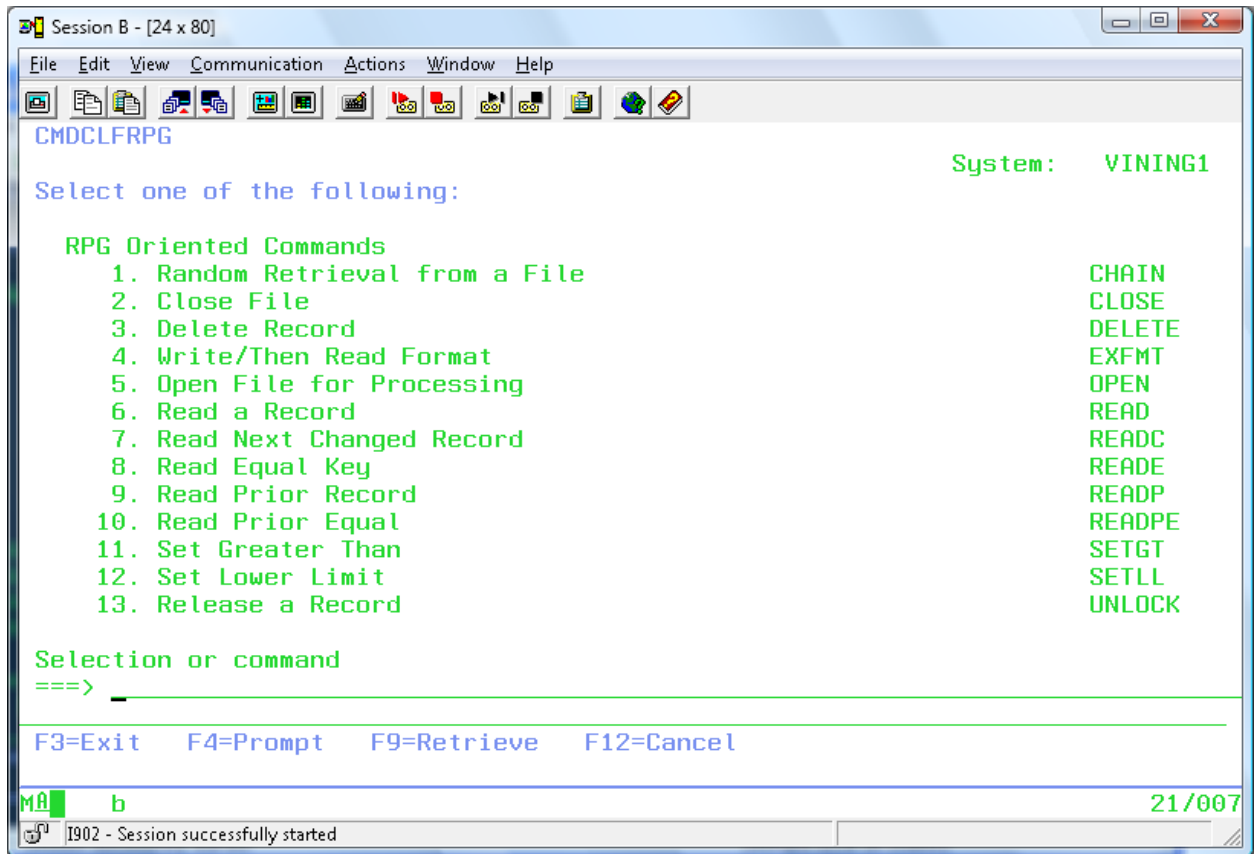
This chapter discusses how to develop, compile, and run a CLF application program. This information is intended primarily as guidance to the CLF developer and represents a Programmers Guide level of information. CLF commands are discussed but not covered in detail. CLF command details at the Reference Manual level can be found in the online help provided with each command. The command help can be accessed on your System i or via a browser at <http://www.brucevining.com>.

A CLF application program is a conventional CL program or procedure in that the program is developed using a set of source statements that consist entirely of CL commands and these source statements are then compiled. The CLF program source can be written using your source editor of choice, compiled using either the CLF precompiler commands or the IBM provided CRT commands depending on the CLF functions you are using in the program, and then run. Running the program can be done by directly calling the program or defining the CLF program as the command processing program (CPP) of a user command.

You can find all of the CLF RPG-like commands by entering the command

```
GO CMDCLFRPG
```

The initial page of the CMDCLFRPG menu looks like:



These RPG-like commands are provided for those developers who may already be familiar with RPG file support. These commands in many cases provide a short hand interface to CLF file support. For instance the RPG-like command

```
Chain KeyList(&EmpNbr) FileID(VC2EMP)
```

randomly reads a record from the VC2EMP database file where the record key is equal to the value of CL variable &EMPNO. This provides the same function as the CL command

```
ReadRcdCLF FileID(VC2EMP) Type(*Key) KeyRel(*EQ) +
  KeyList(&EmpNbr)
```

Both commands support two positional parameters so they could also be coded as

```
Chain &EmpNbr VC2EMP
```

and

```
ReadRcdCLF VC2EMP *Key KeyRel(*EQ) KeyList(&EmpNbr)
```

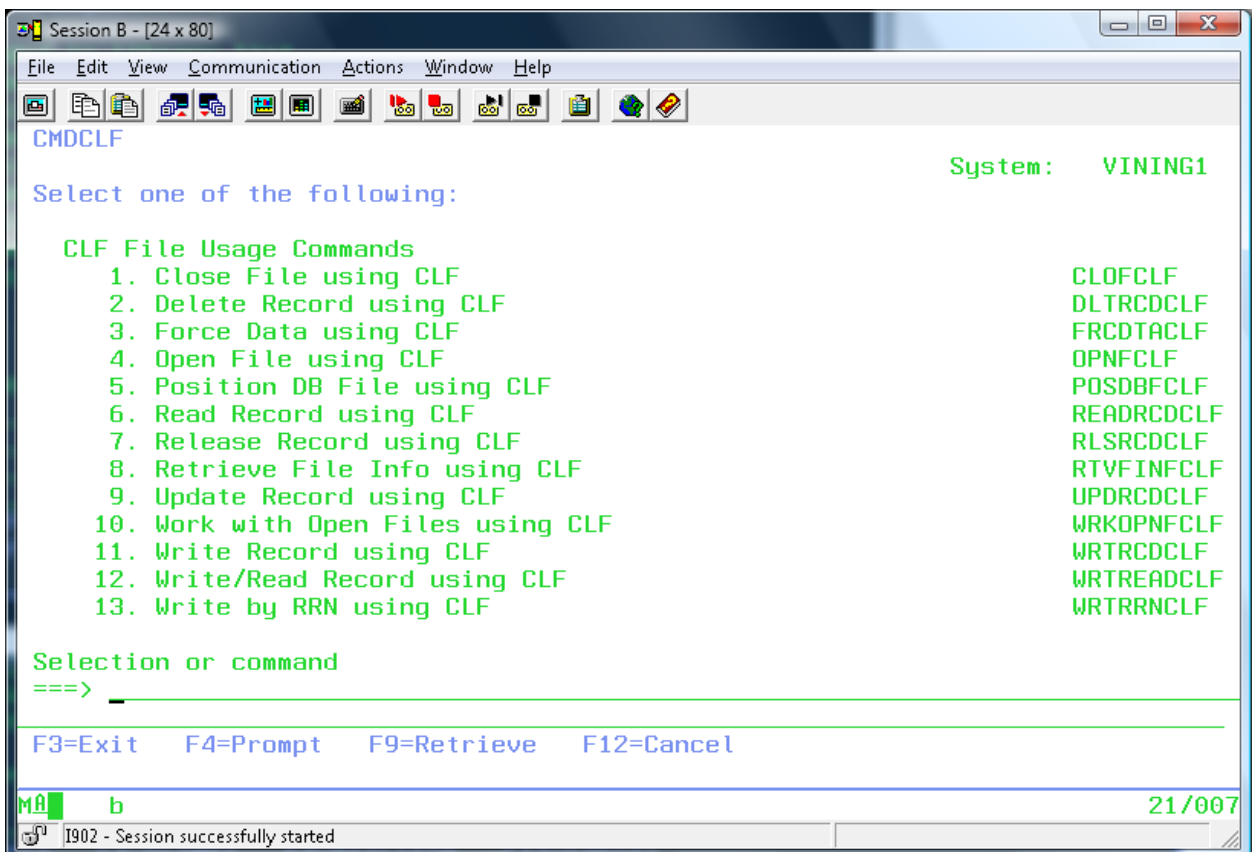
In the case of the RPG CHAIN command most any developer familiar with free-form RPG should readily recognize the function that is being performed. The READRCDF command has quite a few additional parameters that need to be supplied. This is due to the READRCDF command implementing not only the CHAIN command but also READ, READC, READE, READP, and a few more functions that have no RPG read operation equivalent.

These RPG-like commands are only available when using the precompiler. You can freely inter mix these RPG-like commands with the more conventional, in terms of syntax, CLF CL commands in any fashion you like.

By using the Page Down key you can find additional RPG-like commands and also a menu option to take you to the CLF CL syntax oriented menu CMDCLF. You can also access this menu directly using the command

```
GO CMDCLF
```

The initial page of the CMDCLF menu looks like:



The menu is organized with the frequently used CLF commands that are related to file usage shown first. These commands can be used by a CL application program regardless of whether or not you are using the CLF precompiler and/or CLF generation tool commands.

By using the Page Down key you next see the CLF commands that are only available when you are using the CLF precompiler. The precompiler related commands shown provide the ability to compile your CL source, declare and/or include files as part of the compilation, and utilize enhanced database capabilities such as null fields and variable-length fields.

Paging Down further with the CMDCLF menu you also find CLF generation tool commands that provide file declaration capabilities that can be used independent of the precompiler. The generation tool commands require that you have installed option 2 of the CLF product.

Following that you will also find infrequently used CLF commands related to device acquisition and release. The device-related commands can be used regardless of whether or not you are using the CLF precompiler and generation tools.

After these commands is also a menu option to return you to the CLF RPG oriented menu CMDCLFRPG.

## GETTING STARTED

How many times have you been developing a CL application program and found the need to access, or update, information in a file? This is a fairly common occurrence and in the past could have a negative impact on your productivity as you end up having to write a database access program in RPG; defining what parameters need to be exchanged between your CL program and your RPG database access program; coding up the parameters; testing the two programs; remembering to note that changes to the database may require changes to the CL and/or RPG program, and so on. It would be easier all around if you could simply perform the database operation directly out of the CL program. With CLF you have full access to the database of the system directly from your CL application program.

We'll start with a simple CL program that needs to randomly read a record and retrieve the value of specific fields within that record. The examples in this manual will be using the CLF precompiler and RPG-like command syntax.

For the sample application we will have a CL program that, based on the user requesting a report and the name of the printer file used to create the report, needs to:

1. set the proper number of copies for the report

## 2. route the report to the correct output queue

A traditional CL program might retrieve the current user and accomplish this task by running through a set of 'if' or 'when' checks as in:

```
RtvJobA    CurUser(&User)
If         Cond(&User = 'ABLE') Then(OvrPrtF USRLIST Copies(2) +
        OutQ(ABC))
If         Cond(&User = 'ADAMS') Then(OvrPrtF USRLIST Copies(1) +
        OutQ(PGMR))
If         Cond(&User = 'BAKER') Then(OvrPrtF USRLIST Copies(1) +
        OutQ(ABC))
. . .
If         Cond(&User = 'ZIMMER') Then(OvrPrtF USRLIST Copies(3) +
        OutQ(ZIMMER))

Or
RtvJobA    CurUser(&User)
Select
    When Cond(&User = 'ABLE') Then( +
        OvrPrtF USRLIST Copies(2) OutQ(ABC))
    When Cond(&User = 'ADAMS') Then( +
        OvrPrtF USRLIST Copies(1) OutQ(PGMR))
    When Cond(&User = 'BAKER') Then( +
        OvrPrtF USRLIST Copies(1) OutQ(ABC))
    . . .
    When Cond(&User = 'ZIMMER') Then( +
        OvrPrtF USRLIST Copies(1) OutQ(ZIMMER))
    OtherWise Cmd(Do)
        /* Error handling for unknown user */
        EndDo
EndSelect
```

Figure 4. 1 - One traditional approach

But this type of approach leads to ongoing program maintenance as users are added and removed from the system. A better approach would be to have a keyed database file that contains user names, printer file names, number of copies, and the correct output queue for the user. To enable random access to the printer file information the key might be defined as using the user name and the printer file name. Figure 4.2 shows the definition for one possible database, the Print Control (PRTCTL) file.

```
.....A.....T.Name+++++RLen++TDpB.....Functions+++++
                                         UNIQUE
R CTLRCD
  USRPRF      10      TEXT('User Profile')
  PRTF        10      TEXT('Printer file')
```

```

        OUTQ          10          TEXT('Output Queue')
        COPIES        3 0        TEXT('Number of Copies')
K  USRPRF
K  PRTF

```

Figure 4. 2 - Definition of PRTCTL file

If you have installed option 3 of CLF you will find the source for the PRTCTL physical file in member PRTCTL of the source file VC2CLF/QDDSSRC. If you do not have this option installed it is strongly recommended that you install it before continuing. The source code for all subsequent example programs and objects is provided with option 3, CLF Sample Programs.

To create the PRTCTL file into QTEMP you can use the command

```
CRTPF FILE(QTEMP/PRTCTL) SRCFILE(VC2CLF/QDDSSRC)
```

Now the question is, how might the CL program read the appropriate information for a given user?

Traditionally you might declare the file with the DCLF command and then sequentially read the file until the correct user record and printer file is found, run the correct Override with Printer File (OVRPRTF) command, and then run the report. If no record for the user is found, you would assign some default values for the number of copies and output queue. This approach, potentially reading every record in the PRTCTL file, is not overly efficient.

You might try using the Override with Database File (OVRDBF) command and the POSITION keyword in order to avoid potentially reading every record. But unless all of your printer file names are exactly ten characters in length (no trailing blanks) you will find that you end up with a rather complex program for a very simple task. And this is for the simple PRTCTL file where both keys are character based.

Another traditional approach would be to write a RPG program to randomly read the PRTCTL record. You could call the program with four parameters: the user profile value that you pass to the RPG program, the printer file name value (that you pass to the RPG program), the number of copies (that the RPG program returns to the CL program), and the output queue (that the RPG program also returns to the CL program). You would also need to decide what program sets default values in case no record is found. This approach is also quite workable, though it does mean writing another program, some level of design, and can be time consuming if there are many database fields that the CL application program needs access to (though our example only requires two fields – OUTQ and COPIES).

## RANDOMLY READ A RECORD BY KEY

The CLF approach would be to simply read the record directly from the CL program. The following program will read the appropriate PRTCTL record for printer file USRLIST and then assign the correct number of copies and output queue. If no PRTCTL record is found for the user the program will use default values of one copy and an output queue named DEFAULT.

```
Pgm
(A) File      FileID(PRTCTL)
   Dcl       Var(&RNF) Type(*Lgl)

   RtvJobA   CurUser(&UsrPrf)

(B) Open     PRTCTL AccMth(*Key)
(C) Chain    (&UsrPrf 'USRLIST') PRTCTL RcdNotFnd(&RNF)
   If       Cond(&RNF) Then(OvrPrtF File(USRLIST) +
      OutQ(DEFAULT) Copies(1))
   Else     Cmd(OvrPrtF File(USRLIST) OutQ(&OutQ) +
      Copies(&Copies))

(D) Close    PRTCTL

/* Call the program that runs report USRLIST */

DltOvr      File(USRLIST)

EndPgm
```

Figure 4.3 - Randomly reading a record from the PRTCTL file using the CLF precompiler

As with any CL program the first thing we need to do is declare our CL variables. CLF does not change this.

As there is no RPG operation code to declare a file, and a command with a name such as F\_SPEC did not seem appropriate, the File Description Specification (FILE) command, shown at (A) of Figure 4.3, is used to declare a file. The FILE command will cause the precompiler to extract file, record format, and field information from the file named PRTCTL. The precompiler will then generate Declare CL Variable (DCL) commands that define the fields and record formats associated with the PRTCTL file. The precompiler will actually do much more than this when a FILE command is encountered, but for now we're only concerned with the field definition function. There are several other parameters defined for the FILE command and we are taking default values for these additional parameters. If you are interested in reviewing these other parameters you

can prompt the FILE command and then use command key 1 to access the help text for the command. All CLF commands come with extensive online help text.

You may notice a few items that you normally associate with the RPG F-spec but are not on the FILE command. There is for instance no keyword that is equivalent to column 34 (Record Address Type), better known as 'K' or ' ' (blank), to indicate whether or not you want to use keys when accessing a file. With CLF the access method (by key or by relative record number) is specified on the Open File for Processing (OPEN) command, which is another reason for not naming the FILE command F\_SPEC. The access method could have been made a function of the FILE command but it seems more like a characteristic of the file open than a static characteristic of the file you want to use. Later in this chapter you will see that associating the access method with the OPEN of a file, rather than the declaration of the file, can provide some new flexibility in how you develop an application program.

Following the FILE command we also define a logical variable named &RNF for Record Not Found. CLF provides an alternative way to define this variable but again, for now, we'll defer that discussion.

It is worth pointing out though that you can intermix CLF declare commands with IBM provided declare commands. In Figure 4.3 you could just as easily have declared the &RNF CL variable before using the FILE command. The only requirement is that, as with traditional CL programs, all declares follow the Program (PGM) command and precede any executable commands in the program. There are some special capabilities discussed later in Chapter 5. Files, Record Formats, and Fields that are related to the order of declare statements, but these features are not needed by the current program.

After the declare statements the program retrieves the current user profile for the job using the Retrieve Job Attributes (RTVJOBA) command. The CL variable &USRPRF is used to hold the name. The CL variable &USRPRF is not explicitly declared in the program by you with a DCL, but was implicitly declared due to the first field of the PRTCTL database file being named USRPRF (see Figure 4. 2 - Definition of PRTCTL file). It would not be an error if you had defined a &USRPRF CL variable as TYPE(\*CHAR) and LEN(10), but you didn't have to.

At (B) in Figure 4.3 the program opens the PRTCTL file using the Open File for Processing (OPEN) command. As with the previous FILE command, we are taking default values for many parameters that are not shown in the sample program. The OPEN command is opening the file PRTCTL and indicating that keys, rather than relative record numbers, will be used in accessing the records found in the PRTCTL file.

The program now attempts at (C) to randomly read a record from the PRTCTL file where the record key is equal to the value found in the &USRPRF variable and the constant 'USRLIST' (USRLIST is the name of the \*PRTF report that is about to be created). If no record with the specified key value is found the logical variable &RNF is

to be set to true ('1') due to the use of keyword RcdNotFnd with logical variable &RNF. Otherwise the variable &RNF is to be set to false ('0'). If you are familiar with free form RPG IV you probably didn't need any assistance in understanding what the CHAIN command is doing. The one possible exception is in realizing that the RcdNotFnd keyword is logically replacing either the specification of an indicator in columns 71 and 72 or your use of the RPG %FOUND built-in. The CHAIN command supports a list of up to ten keyfields and, as you can see, supports key values comprised of both CL variables (&USRPRF) and literal constants ('USRLIST').

Returning to the program, if the condition &RNF is true the program overrides the printer file USRLIST to defaults of 1 copy and an output queue of DEFAULT. Otherwise the program overrides the printer file USRLIST to the number of copies and output queue found in the record read from the PRTCTL file.

Done with the PRTCTL file the program at (D) closes the file using the Close File (CLOSE) command, runs the report program, deletes the USRLIST override, and returns to the programs caller.

The source for this program can be found in member RPG\_PRTCTL of source file VC2CLF/VC2CLSRC. If you have not done so already, create the PRTCTL file of Figure 4.2. This file is used by the RPG\_PRTCTL program and must exist when compiling the program. To create the file into QTEMP you can use the command

```
CRTPF FILE(QTEMP/PRTCTL) SRCFILE(VC2CLF/QDDSSRC)
```

To compile RPG\_PRTCTL into QTEMP as an ILE application you can use the command

```
CRTBNDCLF PGM(QTEMP/RPG_PRTCTL) SRCFILE(VC2CLF/VC2CLSRC)
```

Note that due to the naming similarity of the CLF CRTBNDCLF command to the IBM provided CRTBNDCL command you can replace any references to CRTBNDCLF with the CLFI (CLF ILE) proxy command.

To create the program as an OPM application you could use the command

```
CRTCLFPGM PGM(QTEMP/RPG_PRTCTL) SRCFILE(VC2CLF/VC2CLSRC)
```

As with the CRTBNDCLF command, due to the naming similarity of the CLF CRTCLFPGM command to the IBM provided CRTCLPGM command you can replace any references to CRTCLFPGM with the CLFO (CLF OPM) proxy command.

If you are using CLF during the 30-day trial period you may see informational messages when running the precompiler commands of CLF. These messages, related to the number of days remaining in the trial period, will not appear once you have installed a license key

for the precompiler. One example of when such a message may appear is when you run the CRTBNDCLF or CRTCLFPGM commands referenced above.

That's it! You have successfully customized the running of a report based on the PRTCTL file in a manner that was relatively fast (both to program and to run), will not require program maintenance as we add and remove users to the system (though we will need to maintain the PRTCTL file – more on that shortly), and did not require another program to be written.

Currently calling RPG\_PRTCTL will cause the default values of COPIES(1) and OUTQ(DEFAULT) to be used. This is because there are currently no records in the PRTCTL file, so any user of the program will fall into the “Record Not Found” category.

Now the question is -- how do you get records into the PRTCTL file? Do you need to write a RPG program to write the records? Use a utility such as the Data File Utility (DFU) or Structured Query Language (SQL)? The answer is obviously ‘No’. You can maintain the PRTCTL file using CLF.

You could write an interactive CL program to maintain the file using CLF display file support (see Work with Print Control in Chapter 9 for an actual example), but for now we'll write our own CL commands to write, change, and delete PRTCTL records. The command processing programs (CPPs) for these commands will be CLF-developed CL programs.

## WRITING A RECORD TO A DATABASE FILE

Figure 4.4 shows the source for the command Add Print Control User (ADDPRTUSR).  
Figure 4.5 shows the source for the CPP RPG\_ADDUSR.

```
CMD          PROMPT('Add Print Control User')
PARM        KWD(USRPRF) TYPE(*SNAME) LEN(10) MIN(1) +
            PROMPT('User profile')
PARM        KWD(PRTF) TYPE(*SNAME) LEN(10) MIN(1) +
            PROMPT('Printer file')
PARM        KWD(OUTQ) TYPE(*NAME) LEN(10) DFT(Default) +
            PROMPT('Output queue')
PARM        KWD(COPIES) TYPE(*DEC) LEN(3 0) RSTD(*NO) +
            DFT(1) RANGE(1 255) PROMPT('Copies')
```

Figure 4.4 - Command definition for ADDPRTUSR

The ADDPRTUSR command defines four parameters. The first parameter, USRPRF, is a required parameter and is the name of the user profile you want to add to the PRTCTL file. The second parameter, PRTF, is also required and is the name of the printer file you want to add to the PRTCTL file for the user identified by the USRPRF parameter. The third parameter, OUTQ, is optional and defines the output queue to be used for reports created using the specified printer file. The default output queue name is DEFAULT. The fourth parameter, COPIES, is optional and defines the number of copies to be printed for the report. The valid range for COPIES is 1 through 255 copies with a default of 1 copy.

The source for ADDPRTUSR can be found in member ADDPRTUSR of source file VC2CLF/QCMDSRC. To create this command into the QTEMP library you can use the command

```
CRTCMD CMD(QTEMP/ADDPRTUSR) PGM(QTEMP/RPG_ADDUSR) +
      SRCFILE(VC2CLF/QCMDSRC)
```

Though the CPP RPG\_ADDUSR does not exist yet the command will still create successfully.

```
/* **** */
/* This program is the CPP for the ADDPRTUSR */
/* command. */
/* **** */

Pgm          Parm(&UsrPrf &PrtF &OutQ &Copies)
```

```

/*****/
/* Declare the PRTCTL file and CLF related */
/* indicators */
/*****/

(A) File      FileID(PRTCTL)

(B) Inds     CLFInd(*Yes)

/*****/
/* Open PRTCTL for keyed access and attempt to */
/* read an entry for the specified user */
/*****/

(C) Open     PRTCTL Usage(*Both) AccMth(*Key)

Chain       (&UsrPrf &PrtF) PRTCTL RcdNotFnd(&RNF)

/*****/
/* If no record is found then write a new */
/* record based on the parameters passed in */
/*****/

If          Cond(&RNF) Then(Do)

(D)         Write PRTCTL

           SndPgmMsg Msg('Report' *BCat &PrtF *BCat +
           'successfully added for user' *BCat +
           &UsrPrf *TCat '.')
           EndDo

/*****/
/* If a record is found then return an error */
/* message and point the user to the CHGPRTUSR */
/* command */
/*****/

Else       Cmd(SndPgmMsg Msg('Report' *BCat +
           &PrtF *BCat +
           'is currently defined for user' *BCat +
           &UsrPrf *TCat '. Use the CHGPRTUSR +
           command to change the existing entry.'))

/*****/
/* Close the PRTCTL file and return to the user*/
/*****/

Close     PRTCTL
EndPgm

```

Figure 4. 5 - The RPG\_ADDUSR command processing program for ADDPRTUSR

The RPG\_ADDUSR command processing program defines four parameters. The first two parameters, &USRPRF and &PRTF, represent the required USRPRF and PRTF keyword parameters of the ADDPRTUSR command. The third and fourth parameters, &OUTQ and &COPIES, correspond to the optional OUTQ and COPIES keywords of ADDPRTUSR respectively.

At (A) the program declares the PRTCTL file using the FILE command previously introduced in Figure 4.3. As the fields of the PRTCTL file have the same name as the parameters being passed to the program, and the file is being declared with the default of FLDSTG(\*AUTO), there is no need for you to DCL the parameters themselves. Following this, at (B), RPG\_ADDUSR uses another CLF command – Indicator Specification (INDS).

In Figure 4.3 you declared the logical variable &RNF but it was mentioned that CLF also provides a way to declare commonly used indicators. The INDS command is that way. One parameter of the INDS command is the keyword CLFIND. When CLFIND(\*YES) is specified the CLF precompiler will automatically declare three indicators for your use. One of these indicators, &RNF, is intended for use in Record Not Found situations. The other two indicators, that will not be used by RPG\_ADDUSR, are &ERR for error conditions and &EOF for end of file conditions when reading a file.

With the CL variable declares done the program now opens the PRTCTL file using the OPEN command. Here a new keyword has been specified. In Figure 4.3 the USAGE keyword was not used. This keyword was not necessary in the earlier example as the default file usage is to allow input operations and all RPG\_PRTCTL needed to do was read records from the PRTCTL file. RPG\_ADDUSR on the other hand needs to write new records and, as coded, also to read records (in order to confirm that you are not trying to add a record for a user and report that currently exists). Because of this requirement you specify USAGE(\*BOTH) at (C) of Figure 4.5. USAGE(\*BOTH) allows you to read, write, update, and delete records within a file.

After opening the file, RPG\_ADDUSR attempts to read the PRTCTL record with a key value equal to the &USRPRF and &PRTF values passed as parameters one and two. As with RPG\_PRTCTL, the logical variable &RNF will be set to true if no record is found, false if a record is found.

If no record is found RPG\_ADDUSR, at (D), writes a new record to the PRTCTL file using the Create New Records (WRITE) command and sends a message to the user indicating that the report for the user was successfully added.

If a record is found RPG\_ADDUSR sends a message to the user indicating that the report is already defined for the user specified and that the Change Print Control User (CHGPRTUSR) command should be used.

Regardless of whether or not a record was added to the PRTCTL file, RPG\_ADDUSR then closes the file using the CLOSE command and returns to the user.

The source for RPG\_ADDUSR can be found in member RPG\_ADDUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_ADDUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

To add the user BAKER and printer file USRLIST to the PRTCTL file specifying a default output queue of BAKERPRT and 2 copies of the report you would use the command

```
ADDPRTUSR USRPRF(BAKER) PRPF(USRLIST) +  
          OUTQ(BAKERPRT) COPIES(2)
```

And in case you are wondering, there is no requirement that the RPG\_ADDUSR program first attempt to read a record from the PRTCTL file prior to writing a record. The program could be written to simply write the record. The PRTCTL file however, per Figure 4.2, is defined to support only UNIQUE keys. So if you were to try and write a duplicate record (one where a record already exists with the key fields of user name and report name set to the same values as the record you are writing) an error would be encountered. To detect this error you could either specify the ERR keyword with the WRITE command or use a MONMSG for message VC2501E - Write to file ID PRTCTL not successful. If an error occurs you could then send a message to the user indicating that the record was not successfully added. Unfortunately, knowing that an error was encountered is not the same as knowing what the error was. The WRITE command may have failed due to a duplicate key, it may have failed due to the file member being at maximum size, or a number of other reasons. The exact reason can be found in the job log (CPF5026 for a duplicate key, CPF5018 for being at maximum size, etc), but does require some additional coding in the application. See Example using the WRITE command of Chapter 7 if you would like to see what is needed to determine the specific cause of the VC2501E error message. Using the CHAIN approach of RPG\_ADDUSR you can be sure that a failure on the WRITE command indicates a truly unusual situation as opposed to a more common situation such as a duplicate key.

## UPDATING A DATABASE RECORD

Having added report USRLIST for the user BAKER to the PRTCTL file we now need a way to maintain or change the information stored in the database. Figure 4.6 shows the source for the command Change Print Control User (CHGPRTUSR). Figure 4.7 shows the source for the CPP RPG\_CHGUSR.

```
CMD          PROMPT('Change Print Control User')
PARM        KWD(USRPRF) TYPE(*SNAME) LEN(10) MIN(1) +
            PROMPT('User profile')
PARM        KWD(PRTF) TYPE(*SNAME) LEN(10) MIN(1) +
            PROMPT('Printer file')
PARM        KWD(OUTQ) TYPE(*NAME) LEN(10) DFT(*SAME) +
            SPCVAL((*SAME *SAME)) PROMPT('Output queue')
PARM        KWD(COPIES) TYPE(*DEC) LEN(3 0) RSTD(*NO) +
            DFT(*SAME) RANGE(1 255) SPCVAL((*SAME +
            -1)) PROMPT('Copies')
```

Figure 4.6 - Command definition for CHGPRTUSR

The CHGPRTUSR command is quite similar to the ADDPRTUSR command of Figure 4.4. The only changes are related to the defaults for OUTQ and COPIES. Both have been changed to the special value \*SAME. In the case of the COPIES keyword we also map the special value \*SAME to the value of -1 as COPIES is defined as a numeric value.

The source for CHGPRTUSR can be found in member CHGPRTUSR of source file VC2CLF/QCMDSRC. To create the command into the QTEMP library you can use the command

```
CRTCMD CMD(QTEMP/CHGPRTUSR) PGM(QTEMP/RPG_CHGUSR) +
        SRCFILE(VC2CLF/QCMDSRC)
```

The CHGPRTUSR command could have been created with a prompt override program (POP), written of course using CLF, to replace the \*SAME special values for the OUTQ and COPIES parameters with the actual values found in the PRTCTL record. But that would not have demonstrated anything new to you -- just how to read a record, a topic you are already familiar with from Figures 4.3 and 4.5.

```
/* **** */
/* This program is the CPP for the CHGPRTUSR */
/* command. */
/* **** */
```

```

Pgm          Parm(&UsrPrf &PrtF &OutQ_In &Copies_In)

/*****
/* Declare the program parameters */
*****/

Dcl          Var(&OutQ_In)   Type(*Char) Len(10)
Dcl          Var(&Copies_In) Type(*Dec)  Len(3 0)

/*****
/* Declare the PRTCTL file */
*****/

File          FileID(PRTCTL)

/*****
/* Open PRTCTL for keyed access and attempt to */
/* read an entry for the specified user */
*****/

Open          FileID(PRTCTL) Usage(*Both) AccMth(*Key)

Chain         (&UsrPrf &PrtF) PRTCTL

/*****
/* If no record is found then: */
/* Return an error message */
/* Point the user to the ADDPRTUSR command */
/* Close the PRTCTL file */
/* Return */
*****/

MonMsg        MsgID(VC2501B) Exec(Do)

                SndPgmMsg Msg('Report' *BCat &PrtF +
                *BCat 'for user' *BCat &UsrPrf +
                *BCat 'not found in PRTCTL. Use +
                the ADDPRTUSR command to change the +
                existing entry.')
```

```

                Close PRTCTL
                Return
                EndDo

/*****
/* If a record is found then update the record */
/* to the new values. Leave the current */
/* values if *SAME specified for the parameter */
*****/

If            Cond(&OutQ_In *NE *Same) Then( +
                ChgVar Var(&OutQ) Value(&OutQ_In))

If            Cond(&Copies_In *NE -1) Then( +

```

```

                                ChgVar Var(&Copies) Value(&Copies_In))

Update      PRTCTL

SndPgmMsg  Msg('Report' *BCat &PrtF *BCat +
              'for user' *BCat &UsrPrf *BCat +
              'successfully changed.')
```

Close PRTCTL  
EndPgm

Figure 4. 7 - The RPG\_CHGUSR command processing program for CHGPRTUSR

As with the RPG\_ADDUSR program RPG\_CHGUSR starts by defining the four parameters passed to the program and declaring the PRTCTL database file using the FILE command. But unlike RPG\_ADDUSR, where you did not explicitly define any of the parameters passed, RPG\_CHGUSR does declare the two parameters &OUTQ\_IN and &COPIES\_IN. This is necessary as you need to preserve these values after reading the PRTCTL record identified by the key values &USRPRF and &PRTF. If you didn't declare different CL variable names for these two parameters then the CHAIN command, when successful, would replace the command parameter values with the database record values.

Another change from RPG\_ADDUSR, where we used the INDS command to declare the &RNF logical variable, and RPG\_PRTCTL, where we explicitly declared the &RNF logical variable, is that the RPG\_CHGUSR program does not declare the variable &RNF at all. Rather than using the RCDNOTFND keyword of the CHAIN command, as was done in the earlier examples, RPG\_CHGUSR will demonstrate another way to detect when a record is not found.

After opening the file using the OPEN command RPG\_CHGUSR attempts at (A) of Figure 4.7 to read a record from PRTCTL where the record key is equal to the &USRPRF and &PRTF parameter values. But note that the RCDNOTFND keyword is not being used. When this keyword is not used CLF will send escape messages to the application program to indicate that the requested record was not found. The help text for the RCDNOTFND keyword documents that message VC2501B (Record not found in file &1) will be sent, so following the CHAIN command is a Monitor Message (MONMSG) command that monitors for VC2501B. This is shown at (B). If this message is received RPG\_CHGUSR sends a message to the user indicating that the specified user report is not in the PRTCTL file and that the user should run the ADDPRTUSR command. RPG\_CHGUSR then closes the PRTCTL file and returns.

If message VC2501B was not received RPG\_CHGUSR uses the Modify Existing Record (UPDATE) command at (C) to update the user's record after checking to see what fields should be modified. It is this check that requires that you define the Output queue and Copies parameters using names that are different than the field names used in the

PRTCTL record format. The program then sends a message indicating the change was successful, closes the PRTCTL file, and returns.

Most CLF commands have one or more keywords that are associated with various forms of feedback. These keywords are Record Not Found (RCDNOTFND), Error Found (ERR), and End of File (EOF). You have the choice when writing an application program using CLF whether you want to use CL variables or MONMSG to handle feedback related to these situations. You can, within the same program or even with the same command, intermix these two feedback mechanisms in whatever manner is appropriate for your needs.

Unlike the RPG\_ADDUSR program, where reading a record prior to writing a new record is optional, the RPG\_CHGUSR program must read the record to be updated prior to using the UPDATE command to update the record. The UPDATE command will update the record that is last read from the file and that is locked for update.

The source for RPG\_CHGUSR can be found in member RPG\_CHGUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_CHGUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

To change the number of copies user BAKER gets of the report USRLIST from 2 to 5 you use the command

```
CHGPRTUSR USRPRF(BAKER) PRTF(USRLIST) COPIES(5)
```

## DELETING ONE OR MORE DATABASE RECORDS

Having added and changed printer file USRLIST for BAKER, it's time to now delete Baker's record from the PRTCTL file. Figure 4.8 shows the source for the command Remove Print Control User (RMVPRTUSR). Figure 4.9 shows the source for the CPP RPG\_RMVUSR.

```
CMD          PROMPT('Remove Print Control User')
PARM        KWD(USRPRF) TYPE(*SNAME) LEN(10) MIN(1) +
            PROMPT('User profile')
PARM        KWD(PRTF) TYPE(*SNAME) LEN(10) SPCVAL((*ALL +
            *ALL)) MIN(1) PROMPT('Printer file')
```

Figure 4.8 - Command definition for RMVPRTUSR

For the RMVPRTUSR command we only need the two parameters USRPRF and PRTF. Both are required parameters as they are necessary to identify the PRTCTL entry that is to be removed. As it is likely that a given user may have multiple entries in the PRTCTL file, one for each report they might run, RMVPRTUSR also defines the special value \*ALL for the PRTF keyword. When \*ALL is used the CPP will delete all records for the specified user name. If a specific PRTF is identified only that specific record will be deleted.

The source for RMVPRTUSR can be found in member RMVPRTUSR of source file VC2CLF/QCMD SRC. To create the command into the QTEMP library you can use the command

```
CRTCMD CMD(QTEMP/RMVPRTUSR) PGM(QTEMP/RPG_RMVUSR) +
      SRCFILE(VC2CLF/QCMD SRC)
```

```
/* **** */
/* This program is the CPP for the RMVPRTUSR */
/* command. */
/* **** */

Pgm          Parm(&UsrPrf_In &PrtF_In)

/* **** */
/* Declare the program parameters */
/* **** */

Dcl          Var(&UsrPrf_In) Type(*Char) Len(10)
Dcl          Var(&PrtF_In) Type(*Char) Len(10)
```

```

/*****/
/* Declare the PRTCTL file and CLF related */
/* indicators */
/*****/

File          FileID(PRTCTL)
Inds          CLFInd(*Yes)

/*****/
/* Open PRTCTL for keyed access */
/*****/

Open          FileID(PrtCtl) Usage(*Both) AccMth(*Key)

/*****/
/* Determine if only one record is to be */
/* deleted or if multiple records might be */
/* deleted */
/*****/

(A) If        Cond(&PrtF_In *NE *ALL) Then(Do)

              /*****/
              /* If one, read the record and */
              /* delete if found.  If not found, */
              /* send an error message */
              /*****/

              Chain (&UsrPrf_In &PrtF_In) PRTCTL +
                  RcdNotFnd(&RNF)

              If Cond(*Not &RNF) Then(Do)

(B)           Delete PRTCTL

              SndPgmMsg Msg('Report' *BCat &PrtF +
                  *BCat 'for user' *BCat &UsrPrf *BCat +
                  'successfully deleted.')
              EndDo

              Else Cmd(SndPgmMsg Msg('Report' *BCat +
                  &PrtF_In *BCat 'for user' *BCat +
                  &UsrPrf_In *BCat +
                  'not found in PRTCTL.'))

              EndDo

Else         Cmd(Do)

              /*****/
              /* If multiple records are possible */
              /* attempt to position to the first */
              /*****/

(C)          Setll &UsrPrf_In PRTCTL

```

```

/*****/
/* Attempt to read the first record */
/*****/

(D)  ReadE &UsrPrf_In PRTCTL EOF(&EOF)

/*****/
/* If none found, send error message*/
/*****/

If Cond(&EOF) Then( +
    SndPgmMsg Msg( +
        'No entries found for user' *BCat +
        &UsrPrf_In)

Else Cmd(Do)

/*****/
/* If one or more found, read all */
/* of them, delete them, and send a */
/* message for each one. */
/*****/

        DoWhile (*Not &EOF)
            Delete PRTCTL
            SndPgmMsg Msg('Report' *BCat +
                &PrtF *BCat 'removed.')
            ReadE &UsrPrf_In PRTCTL EOF(&EOF)
            EndDo

/*****/
/* When done with all of them send */
/* a general message saying so */
/*****/

        SndPgmMsg Msg( +
            'All reports removed for user' +
            *BCat &UsrPrf_In)
        EndDo
    EndDo

/*****/
/* Close the PRTCTL file and return to the user*/
/*****/

Close      PRTCTL
EndPgm

```

Figure 4. 9 - The RPG\_RMVUSR command processing program for RMPRTUSR

As with the earlier examples, RPG\_RMVUSR starts by declaring the parameters being used, the database file PRTCTL, the logical variables for feedback from the CLF commands, and opening the PRTCTL file.

At (A) of Figure 4.9 RPG\_RMVUSR checks the &PRTF\_IN parameter to determine if the special value \*ALL is being used. If not the program attempts to read the record with key value of &USRPRF\_IN for the user name and &PRTF\_IN for the report name. If the record is successfully read the record is deleted using the Delete Record (DELETE) command at (B), an appropriate message is sent, the PRTCTL file is closed, and the program returns. If the record is not found an error message is sent, the PRTCTL file is closed, and the program returns. Based on what you learned with the previous RPG\_PRTCTL, RPG\_ADDUSR, and RPG\_CHGUSR example programs this part of RPG\_RMVUSR should be fairly straight-forward to understand.

Several new features however are introduced if the special value \*ALL is used for the PRTF keyword of RMVPRUSR. In this case, at (C) RPG\_RMVUSR uses the command Set Lower Limit (SETLL) to position the file to the first record with a key equal (or greater than if there is no equal key in the file) to the value of the &USRPRF\_IN parameter, followed by the Read Equal Key (READE) command. If, with the READE command, there is no record with the specified key value then logical variable &EOF is to be set to true indicating that logical end-of-file has been reached. If a record is found with the specified key value then logical variable &EOF is to be set to false. As an aside, you might notice that while the PRTCTL file is defined as having two key fields you can use CLF commands to process records in the file using fewer keys. This is true for both the SETLL and the CHAIN commands, plus several others.

After running the READE command the program checks the &EOF variable. If this variable is true a message indicating that no entries exist for the specified user is sent. If this variable is false, indicating that one or more records with the specified key value do exist, RPG\_RMVUSR performs additional processing.

Upon verifying that records matching the &USRPRF\_IN value do exist, RPG\_RMVUSR enters a DOWHILE loop that is conditioned by End of File (logical variable &EOF) not being true. Within the DOWHILE loop RPG\_RMVUSR deletes each record at (E) using the Delete Record (DELETE) command, sends a messages identifying the report that was removed, attempts to read the next record with a matching key, and returns to the initial DOWHILE (\*Not &EOF) compare. This DOWHILE cycle will continue for as long as more records are found with the specified &USRPRF\_IN parameter value. When all records for &USRPRF\_IN have been deleted the DOWHILE loop ends, a message is sent confirming that all records have been removed, the program closes the PRTCTL file, and returns.

It's worth noting that the SETLL and READE commands are being used with one key value (&USRPRF\_IN when the \*ALL special value is used for PRTF) while CHAIN uses two key values (&USRPRF\_IN and &PRTF\_IN when a specific report is identified

with the PRTF keyword). You decide how many keys are to be used for each file related command when processing your database. Though not shown in this example the KEYSTRUCT keyword allows you to dynamically decide how many key fields to use at run-time (as opposed to the RPG\_RMVUSR example where the number of key fields is being specified at compile time).

Similar to the RPG\_CHGUSR program, the RPG\_RMVUSR program must read the record from the PRTCTL file before it can delete the record. The DELETE command will delete the record that is last read from the file and that is locked for update.

The source for RPG\_RMVUSR can be found in member RPG\_RMVUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_RMVUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

To test the RMVPRTUSR command run the following commands:

```
ADDPRTUSR USRPRF(ABLE) PRTF(USRLIST) OUTQ(ABLEPRT) +  
          COPIES(3)  
ADDPRTUSR USRPRF(BAKER) PRTF(REPORT1) OUTQ(BAKERPRT)  
ADDPRTUSR USRPRF(BAKER) PRTF(REPORT2) OUTQ(BAKERPRT)  
ADDPRTUSR USRPRF(BAKER) PRTF(REPORT3) COPIES(10)  
ADDPRTUSR USRPRF(COOK) PRTF(REPORT2)
```

Using the command DSPPFM PRTCTL you should see five (or six if you previously added report USRLIST to BAKER back when working with the ADDPRTUSR command) records.

Now use the command

```
RMVPRTUSR USRPRF(BAKER) PRTF(*ALL)
```

Re-running the DSPPFM PRTCTL command you should now only see the two records associated with ABLE and COOK. All of BAKER's entries should have been deleted from the file.

To also delete ABLE's entry for the USRLIST report you can use

```
RMVPRTUSR USRPRF(ABLE) PRTF(USRLIST)
```

## USING A DISPLAY FILE

Having seen how to read, write, update, and delete database records from a CL program let's look at what is required to display a PRTCTL record from an interactive application. Figure 4.10 shows the source for the display file PRTCTLINQ. Figure 4.11 shows the source for the program RPG\_SHWUSR.

```

                                     CA03(03)
R PROMPT
                                     2  4DATE
                                     EDTCDE(Y)
                                     2  70TIME
                                     3  30'PRTCTL Inquiry'
                                     6  11'Enter user name:'
USRPRF  R      I  6  34REFFLD(USRPRF PRTCTL)
                                     8  11'Enter report name:'
PRTF    R      I  8  34REFFLD(PRTF PRTCTL)
50
                                     ERRMSG(+
                                     'No entry found' 50)
                                     23  6'F3=Exit'
R DISPLAY
                                     2  4DATE
                                     EDTCDE(Y)
                                     2  70TIME
                                     3  30'PRTCTL Inquiry'
USRPRF  R      O  8   6'Name:      '
                                     8  23REFFLD(USRPRF PRTCTL)
PRTF    R      O  9   6'Report:    '
                                     9  23REFFLD(PRTF PRTCTL)
OUTQ    R      O  11  6'Output queue:'
                                     11 23REFFLD(OUTQ PRTCTL)
COPIES  R      O  12  6'Copies:    '
                                     12 23REFFLD(COPIES PRTCTL)
                                     23  6'F3=Exit'
```

Figure 4. 10 - Definition for display file PRTCTLINQ

There are two record formats within the PRTCTLINQ display file. The first record format, PROMPT, prompts the user for a user profile name and a report name.

The second record format, DISPLAY, is used when a record is found in PRTCTL for the user name and report requested on the PROMPT display. DISPLAY is used to display the user profile name, report name, output queue to spool the report to, and the number of copies to be printed for the report. If the Enter key is used with the DISPLAY record format the user is returned to the PROMPT screen to continue with more inquiries. If

command key 3 is used from either record format the program ends. Using command key 3 causes indicator 03 to be set on.

From the PROMPT display, when a record is not found in PRTCTL for the requested user name and report, the user remains on the PROMPT screen. Indicator 50 is used in this case to show an error message indicating that no entry was found for the user and report.

The source for PRTCTLINQ can be found in member PRTCTLINQ of source file VC2CLF/QDDSSRC. To create the display file into QTEMP you can use

```
CRTDSPF FILE (QTEMP/PRTCTLINQ) SRCFILE (VC2CLF/QDDSSRC)
```

```
/******  
/* This program provides an inquiry function */  
/* to the PRTCTL file */  
/******  
  
Pgm  
  
/******  
/* Declare the PRTCTL database file, the */  
/* PRTCTLINQ display file, and CLF indicators */  
/******  
  
File PRTCTL  
File PRTCTLINQ  
Inds CLFInd(*Yes)  
  
/******  
/* Open PRTCTL and PRTCTLINQ */  
/******  
  
Open PRTCTL AccMth(*Key)  
Open PRTCTLINQ Usage(*Both)  
  
/******  
/* So long as the user does not press CF03 */  
/* prompt for the user and report name to show */  
/******  
  
DoUntil Cond(&IN03)  
  
/******  
/* Display the PROMPT display */  
/******  
  
(A) Exfmt RcdFmt(Prompt)  
If Cond(&IN03) Then(Leave)
```

```

(B)          Chain (&UsrPrf &PrtF) PRTCTL RcdNotFnd(&RNF)

                /*****
                /* If no record found, display      */
                /* error message with &IN50         */
                /*****

                If Cond(&RNF) Then( +
                  ChgVar Var(&IN50) Value('1'))

                /*****
                /* If record found, display it      */
                /*****

(C)          Else Cmd( +
                Exfmt RcdFmt(Display))

                EndDo

                /*****
                /* Close our files and return when CF03 used */
                /*****

                Close      PRTCTL
                Close      PRTCTLINQ

                EndPgm

```

Figure 4. 11 - The RPG\_SHWUSR program

As with the earlier examples, RPG\_SHWUSR first declares the files that are to be used using the FILE command, CLF logical variables using the INDS command, and opens the files using the OPEN command. With RPG\_SHWUSR there are two files – the PRTCTL database file and the PRTCTLINQ display file.

After opening the files RPG\_SHWUSR enters into a DOUNTIL loop that will end when logical variable (indicator) &IN03 is set to true or on. &IN03 is set to true when the user presses command key 3 from any of the PRTCTLINQ display formats. Within the DOUNTIL RPG\_SHWUSR at (A) uses the EXFMT command to write and then read the PROMPT record format. If the user presses command key 3 on the PROMPT display then logical variable (indicator) &IN03 is set to true and the program LEAVES the DOUNTIL loop. Upon exiting the DOUNTIL loop the program closes the files and returns.

When &IN03 is false the program at (B) attempts to read a record from PRTCTL, using the CHAIN command, where the record key values are equal to the requested &USRPRF and &PRTF names. If no record is found logical variable &RNF is set to true due to the use of keyword RCDNOTFND.

If &RNF is true then logical variable &IN50 is set to true ('1'). As a reminder, the variable &IN50 is used with the PROMPT record format to display the error message 'No entry found'.

If &IN50 is false, indicating that a record was found for the specified user name and report name, the EXFMT command is used at (C) to write and then read the DISPLAY record format. If the user presses command key 3 from the DISPLAY record format then logical variable &IN03 is set to true. &IN03 being true will cause the program to exit the DOUNTIL loop.

Notice that there is no need for you to move (CHGVAR) the data read from PRTCTL file variables to the variables defined by the PRTCTLINQ DISPLAY record format. The precompiler, when files are declared with the FILE command and FLDSTG(\*AUTO), will automatically share data across files and record formats based on fields having the same name (for instance the field OUTQ existing in both PRTCTL and PRTCTLINQ). Because the field names used in the DISPLAY record format are the same as the field names used in the CTLRCD record format of the PRTCTL file the appropriate data is shown without you having to move the data from one record format to another. This automatic sharing, based on field names, is not true when using CL's multiple file support or CLF's base run-time or generation tools (that is, not using the precompiler). Without the precompiler you need to code CHGVAR commands to move data in a variable such as &OUTQ from the PRTCTL file record format to the &OUTQ field of the PRTCTLINQ file record format.

Getting back to the review of RPG\_SHWUSR, if the user has not pressed command key 3 (logical variable &IN03 is not true) the program re-enters the DOUNTIL loop and the PROMPT record format is shown once again.

The RPG\_SHWUSR program is very simple in nature but hopefully gives you an idea of what is required to develop an interactive application that also works with database files on your system. If you are familiar with developing interactive RPG applications you should find no difficulty moving into a CLF environment.

The source for RPG\_SHWUSR can be found in member RPG\_SHWUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_SHWUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

To run RPG\_SHWUSR you simple use

```
CALL PGM(RPG_SHWUSR)
```

While the RPG\_SHWUSR program is reasonably straight forward it can be simplified in terms of the number of lines of code. At (B) in RPG\_SHWUSR logical variable &RNF is

used to indicate that a PRTCTL record is not found for the requested user and report. When &RNF is true, &IN50 is then set to true in order to display an error message to the user. But there is nothing that stops you from directly using &IN50 with the RCDNOTFND keyword of the CHAIN command. Just because CLF provides standard indicators for feedback doesn't mean you have to use them. You can use those CL variables that make the most sense to you.

Figure 4.12 shows the source for a more streamlined version of RPG\_SHWUSR. The name for this example is RPG\_DSPUSR.

```

/*****
/* This program provides an inquiry function  */
/* to the PRTCTL file                        */
/*****

Pgm

/*****
/* Declare the PRTCTL database file and the  */
/* PRTCTLINQ display file                   */
/*****

File      FileID(PRTCTL)
File      FileID(PRTCTLINQ)

/*****
/* Open PRTCTL and PRTCTLINQ                */
/*****

Open      PRTCTL AccMth(*Key)
Open      PRTCTLINQ Usage(*Both)

/*****
/* So long as the user does not press CF03  */
/* prompt for the user and report name to show */
/*****

DoUntil   Cond(&IN03)

          /*****
          /* Display the PROMPT display          */
          /*****

          ExFmt RcdFmt(Prompt)
          If Cond(&IN03) Then(Leave)

(A)       Chain (&UsrPrf &PrtF) PRTCTL +
          RcdNotFnd(&IN50)

          /*****
          /* If record found, display it      */

```

```

/*****/

If Cond(*Not &IN50) Then( +
    ExFmt RcdFmt(Display))

EndDo

/*****/
/* Close our files and return when CF03 used */
/*****/

Close      PRTCTL
Close      PRTCTLINQ

EndPgm

```

Figure 4. 12 - The RPG\_DSPUSR program

With RPG\_DSPUSR variable &IN50 is used at (A) with the RCDNOTFND keyword of the CHAIN command. As &IN50 will be set to true by the CHAIN command (in the case of a record not being found in the PRTCTL file) the only conditional logic the program needs is to write and read the DISPLAY record format when &IN50 is false. And as logical variable &RNF is no longer being used you can also remove the use of the INDS command from the example.

RPG\_SHWUSR and RPG\_DSPUSR are functionally equivalent. RPG\_SHWUSR is shown initially as you may find it easier to follow the flow of that program example. The explicit IF/THEN/ELSE using variable &RNF assists in clarifying what processing is being done. The COND(\*NOT &IN50) test of RPG\_DSPUSR is not as obvious. But you can elect to use either style (or several others) that fits your needs.

The source for RPG\_DSPUSR can be found in member RPG\_DSPUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_DSPUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

## USING A PRINTER FILE

Figure 4.13 shows the source for the command Print Print Control User (PRTPRUSR).  
Figure 4.14 shows the source for the externally described printer file PRTCLRPT.  
Figure 4.15 shows the source for the program RPG\_PRTUSR.

```
CMD          PROMPT('Print Print Control User')
PARM        KWD(USRPRF) TYPE(*SNAME) LEN(10) MIN(1) +
           PROMPT('User profile')
```

Figure 4. 13 - Command definition for PRTPRUSR

The PRTPRUSR command has one required parameter – the name of the user profile. The source for PRTPRUSR can be found in member PRTPRUSR of source file VC2CLF/QCMDSRC. To create the command into QTEMP use

```
CRTCMD CMD(QTEMP/PRTPRUSR) PGM(QTEMP/RPG_PRTUSR) +
       SRCFILE(VC2CLF/QCMDSRC)
```

```
      R HEADING                SPACEB(2)
                                4DATE
                                EDTCDE(Y)
                                70TIME
                                SPACEA(2)
                                25'Print Control Listing for'
      USRPRF          10        +1SPACEA(3)
                                5'Report'
                                20'Output Queue'
                                35'Copies'
                                SPACEA(1)

      R DETAIL                  SPACEA(1)
      PRTF            10        5
      OUTQ            10        21
      COPIES          3  0      36EDTCDE(1)

      R ENDING                  SPACEB(1)
                                2'End of Listing'
```

Figure 4. 14 - Definition for printer file PRTCLRPT

In Figure 4.14 three record formats are defined for the PRTCLRPT printer file. The first, HEADING, represents the title and column headings for the report. The second, DETAIL, is printed once for each record in the PRTCLR file that is related to the user name specified with the PRTPRUSR command. Each detail line includes the printer file

used, the output queue for the report, and the number of copies. The third record format, ENDING, prints the single line 'End of Listing' to indicate that all reports in the PRTCTL file for the specified user have been listed.

The source for printer file PRTCTLRPT can be found in member PRTCTLRPT of source file VC2CLF/QDDSSRC. To create the printer file into QTEMP use

```
CRTPRTF FILE (QTEMP/PRTCTLRPT) SRCFILE (VC2CLF/QDDSSRC)
```

```

/*****/
/* This program prints a report listing the      */
/* PRTCTL entries for a given user.            */
/*****/

Pgm          Parm(&UsrPrf)

/*****/
/* Declare the PRTCTL database file, the      */
/* PRTCTLRPG printer file, and CLF indicators */
/*****/

File          FileID(PRTCTL)
File          FileID(PRTCTLRPT)
Inds          CLFInd(*Yes)

/*****/
/* Open PRTCTL and PRTCTLRPT                  */
/*****/

Open          PRTCTL AccMth(*Key)
Open          PRTCTLRPT Usage(*Output)

/*****/
/* Do the report headings                      */
/*****/

(A) Write      RcdFmt(Heading)

/*****/
/* Position to the first record for the user  */
/*****/

(B) Setll      &UsrPrf PRTCTL

/*****/
/* Print all of the records for this user     */
/*****/

(C) ReadE      &UsrPrf PRTCTL EOF(&EOF)

DoWhile       Cond(*Not &EOF)

```

```

(D)          Write RcdFmt(Detail)

(E)          ReadE &UsrPrf PRTCTL EOF(&EOF)

              EndDo

/******
/* Show that the report is complete          */
/******

(F) Write    RcdFmt(Ending)

/******
/* Close our files and return                */
/******

Close       PRTCTL
Close       PRTCLRPT

EndPgm

```

Figure 4. 15 - The RPG\_PRTUSR command processing program for PRTPTUSR

The RPG\_PRTUSR program is conceptually very similar to the RPG\_RMVUSR program of Figure 4. 9 when the PRTF(\*ALL) option is being used. The major difference is that rather than deleting all of the PRTCTL records for the specified user, RPG\_PRTUSR prints all of the records.

RPG\_PRTUSR starts by declaring the PRTCTL and PRTCLRPT files using the FILE command and CLF related indicators using the INDS command. The program then opens the PRTCTL file for input processing and the PRTCLRPT printer file for output processing.

Moving into the actual processing of RPG\_PRTUSR, at (A) the program outputs the headings for the report being created using the Write Record (WRITE) command. This is the same command as we used when writing new records to the PRTCTL database with the RPG\_ADDUSR command. It's just that now you are writing to a printer file rather than a database file.

There is however a change in the keywords that are being used with this WRITE command. In earlier examples with the PRTCTL file the FILEID keyword is used when writing to PRTCTL. In RPG\_PRTUSR the RCDFMT keyword is being used when writing to PRTCLRPT. The reason for this difference is due to the PRTCTL file only having one record format defined while the PRTCLRPT file has three record formats. The RCDFMT keyword defaults to the special value \*ONLY, meaning that there is only one record format defined for the file. For PRTCTL the default of \*ONLY is sufficient.

For PRTCTLRPT, with three record formats, you need to explicitly indicate the record format that is to be used.

Earlier examples using PRTCTL could have used RCDFMT instead of (or in addition to) FILEID, but FILEID is generally easier to remember. In addition FILEID can also be specified positionally if you prefer that style. Due to the use of the precompiler and FILE, you could use any one of the following WRITE commands for PRTCTL:

```
WRITE PRTCTL
WRITE FILEID (PRTCTL)
WRITE RCDFMT (CTLRCD)
WRITE PRTCTL RCDFMT (CTLRCD)
WRITE FILEID (PRTCTL) RCDFMT (CTLRCD)
```

Any of these five commands would have resulted in a record being written to the PRTCTL file. Writing to the PRTCTLRPT file however requires the use of the RCDFMT keyword. In writing to PRTCTLRPT you could use any of these commands:

```
WRITE RCDFMT (HEADING)
WRITE PRTCTLRPT RCDFMT (HEADING)
WRITE FILEID (PRTCTLRPT) RCDFMT (HEADING)
```

Having written the headings for the PRTCTLRPT report, you now position the PRTCTL file at (B) to the first record with a key value greater than or equal to the specified user name (&USRPRF) passed as a parameter to the program. Upon positioning the file to the first record with a key value greater than or equal to the specified user, the program at (C) then reads the next record with a key equal to the user name using the READE command. If no record has a key value equal to &USRPRF the logical variable &EOF is set to true. If a record with a matching key value is found then &EOF is set to false.

The program then enters a DOWHILE loop conditioned by &EOF being false. So long as &EOF is false the program will write a detail line using the WRITE command at (D), read the next record with a key value equal to &USRPRF at (E), and then rerun the DOWHILE loop checking for the current value of &EOF.

When &EOF is true the DOWHILE loop is exited, the last line of the report is written using the WRITE command with the ENDING record format (F), the PRTCTL and PRTCTLRPT files are closed, and the program returns.

The source for RPG\_PRTUSR can be found in member RPG\_PRTUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_PRTUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

To test the program you can use the PRTUSR command. Try it with both a name that is found in the PRTCTL file and a name that does not exist in the PRTCTL file. If you need to add records to PRTCTL you can use the ADDPRTUSR command.

## WHY DEVELOP USING THE CLF PRECOMPILER?

When you are writing a CLF application program you have a choice in terms of the development environment you will be creating the program in. CLF provides a precompiler environment that can greatly simplify your development efforts and provide you with access to database capabilities not available otherwise. A key point is that a CLF application created using the precompiler will run on a system that does not have the precompiler installed. All CLF application capabilities that are enabled by the precompiler are available to all systems that have CLF base run-time support installed. The choice of whether or not to use the precompiler is strictly a development decision, not a run-time decision.

### *Enhanced Productivity*

- If you are familiar with RPG then the most obvious reason is access to free-form RPG-like CLF commands to work with files on the system. Many RPG developers can be productively using CLF in less than a day.
- Less coding – the precompiler will set appropriate values for most CLF input/output related commands. For instance the precompiler can determine and set the appropriate CL variable to use for the Record Buffer (RCDBUF) keyword of the READRCDF command (which is the CL-syntax command for reading records when not using a precompiler). Without the use of the precompiler the developer must explicitly code the RCDBUF keyword
- More flexible support for multiple files – for instance the FILE precompiler support can allow the value of a field to be shared across multiple files and record formats when FLDSTG(\*AUTO) is specified. This can allow a consistent view of a common field such as &CustNbr (Customer number) that might exist in more than one file. You don't have to use CHGVAR commands to explicitly move field values from one file or record format to another.
- More flexible compilation support – for instance the INCLUDE command provided by CLF in V5R4 allows you to include CL source from one or more source members into the current compilation. When directly compiling with CRTBNDCL or CRTCLPGM the IBM provided INCLUDE support does not provide for nesting of INCLUDE commands and is not available unless you are at V6R1 or a later release
- Ease of prototyping – combining the flexibility of CL with the functions of CLF provides for rapid prototyping of database access scenarios. Even if your final application will be written in a language such as RPG or COBOL, you may find that the command prompting, online help, data type flexibility, and run-time

flexibility of CLF allows you to test and analyze possible database approaches much faster than when using other languages.

### ***CLF Commands Not Available Outside the Precompiler Environment***

Some CLF commands are only supported by the precompiler. These commands fall into one or more of the following categories:

- Access to additional database capabilities – for instance using the Retrieve Null Attribute (RTVNULACLF) and Change Null Attribute (CHGNULACLF) commands allow you to test or set the null attribute associated with a database field. Directly compiling with CRTBNDCL or CRTCLPGM provides you with no access to this null attribute
- More flexible access to database capabilities – for instance the Retrieve Variable Field Length (RTVVFLCLF) and Set Variable Field Length (SETVFLCLF) commands allow you to retrieve and set the actual length of a variable length database field. In addition the SETVFLCLF command provides support for calculating the actual length based on a trim character that you specify. Directly compiling with CRTBNDCL and CRTCLPGM requires the use of the %Binary built-in in order to access the variable length field's actual length and for you to calculate a fields actual length yourself
- Avoid bugs in the application – for instance the FILE precompiler support can allow a variable length field to be referenced directly by name. Not using the precompiler and directly compiling a program with CRTBNDCL or CRTCLPGM, where files contain variable length fields and the variable length field is the target (VAR) of a CHGVAR command, can lead to inadvertent corruption of the CL variables value if referenced directly by name.

As this type of bug is subtle (after all a developer wouldn't code an obvious bug!) an example might help. Assume that you have a database defined with the following variable length field using DDS (note that the following example would also apply to a SQL table using a variable length field):

```
      TXTSTR      30000          TEXT('Text String')
                               VARLEN(50)
```

If a standard CL program were to declare this file with the DCLF command, read a record from this file where the value of &TxtStr was a 9-byte text string of 'Some text', and then run the simple command

```
ChgVar      Var(&TxtStr)      Value('$40,000')
```

the value of the CL variable &TxtStr would become corrupted. Without going into the details, the system would now believe that &TxtStr was a 23,540-byte text string with a value of '0,000' followed by 23,535 blank characters. From a CL point of view though, all would appear to be correct. Using the system interactive debug capability for instance, `eval &TxtStr` would show the value to be '\$40,000'.

This corruption might only be local to the CL program as standard CL does not provide the ability to update databases, but a bug has been introduced to your applications. And in the future this corruption could impact your system if &TxtStr were to be passed (or received) as a parameter with another program – for instance an ILE RPG program – that does understand variable length fields and that can update databases, or a CLF program not using the precompiler but equally capable of updating databases. These other programs would not detect an error in &TxtStr, but they would be working with the wrong data.

The precompiler, on the other hand, would by default have avoided this problem with the &TxtStr variable. Using the precompiler

```
ChgVar      Var(&TxtStr)      Value('$40,000')
```

will correctly set the variable &TxtStr to the intended value of '\$40,000' (and not '0,000'). Though, like with ILE RPG and the %len built-in, you would also need to use the Set Variable Field Length (SETVFLCLF) command

```
SetVFLCLF  Var(&TxtStr)
```

to set the length of &TxtStr to the correct 7 byte value. Otherwise the system would believe &TxtStr contained the value '\$40,000' with 2 trailing blanks (due to the original value being nine bytes in length).

To further complicate the problem, a simple

```
ChgVar      Var(&TxtStr)      Value('Some text')
```

by a CL program (not using the precompiler) would also result in corrupted data. But in this case, if passed to an ILE RPG program supporting variable length fields, the error may (or may not depending on how the RPG application is written) result in the message RNX0115 (Length of varying length variable is out of range) message being sent. So now we have a bug that sometimes results in an error message and sometimes doesn't – all depending on the actual data value that is used, even though there is a problem regardless of the actual data value used.

It is due to these types of exposures that IBM, by default, does not allow DCLF to process files containing variable length fields. But variable length fields are

becoming more and more common in application databases and CL, with the CLF precompiler, can handle them.

All of these points, and more, essentially add up to the CLF precompiler providing improved programmer productivity and greater functionality within CL application programs.

## COMPILING A CLF PROGRAM

This section discusses how to compile a CLF program using CL source and a CLF precompiler. A CLF program can be either an Integrated Language Environment (ILE) program or an Original Program Model (OPM) program depending on the command you use when creating the program. The Create Bound CLF Program (CRTBNDCLF) command creates ILE CLF programs using the precompiler. The Create CLF Module (CRTCLFMOD) command creates ILE CLF modules using the precompiler. The Create CLF Program (CRTCLFPGM) command creates OPM programs using the precompiler. The precompiler will run the appropriate IBM provided CRT command as part of normal processing if no errors are detected.

If you are not using the precompiler you can run the IBM provided CRTBNDCL, CRTMODCL, or CRTCLPGM command directly. In this case there is no additional processing performed by CLF.

### *Using the CRTBNDCLF Command*

The Create Bound CLF (CRTBNDCLF) command runs the CLF precompiler. The precompiler processes your CL source prior to having the source statements compiled into a program. As a part of this processing the precompiler analyzes CLF commands, a subset of IBM provided commands, and any commands related to the definition of CL variables. A listing of the source examined and any errors that were detected is generated to help you correct any errors that occur. Because the name of the CRTBNDCLF command is very similar to the IBM provided CRTBNDCL program, you can also use the proxy command CLFI rather than CRTBNDCLF.

The CLF precompiler creates a temporary source member that contains your original source and any expanded source that CLF has added. If no severe errors are found during the analysis phase the precompiler runs the appropriate IBM create commands to compile this expanded source and create your application program. The object description for the created program object will reflect the name of your original source file and source member, not the name of the temporary source member the precompiler creates.

Precompiler processing is dependent on the use of the CLF declare commands FILE, INDS, and DCLFKSCLF. If a file is not declared within the CL program using FILE (or, if using CL syntax DCLFCLF) then no additional processing of CLF commands, using the file, is done.

If no CLF declare commands are found by the precompiler then the precompiler uses the original source member when creating the program. This means that you can elect to always compile using the precompiler command.

The CRTBNDCLF command supports the same keyword parameters and values as the CRTBNDCL command. Many of the keyword values are passed through to the CRTBNDCL command as is. The major exception is related to the SRCFILE and SRCMBR keywords. Due to the precompiler using a temporary source file member the values you specify for these two keywords are changed. The use of a temporary source member also has an impact on the DBGVIEW keyword. The \*SOURCE option for DBGVIEW, though supported as a valid value, is not of practical value as the source member will most likely not exist at the time of attempting to debug the application program. A \*SOURCE view will only be available in the job that ran the precompiler and only so long as there is no additional use of the precompiler. For these reasons it is recommended that you specify DBGVIEW(\*LIST) if you anticipate needing to debug the application at a later time.

The CRTBNDCLF command can be used either interactively or in batch.

### ***Using the CRTCLFMOD Command***

The Create CLF Module (CRTCLFMOD) command runs the CLF precompiler. The precompiler processes your CL source prior to having the source statements compiled into a module. As a part of this processing the precompiler analyzes CLF commands, a subset of IBM provided commands, and any commands related to the definition of CL variables. A listing of the source examined and any errors that were detected is generated to help you correct any errors that occur.

The CLF precompiler creates a temporary source member that contains your original source and any expanded source that CLF has added. If no severe errors are found during the analysis phase the precompiler runs the appropriate IBM create command to compile this expanded source and create your application module. The object description for the created module object will reflect the name of your original source file and source member, not the name of the temporary source member the precompiler creates.

Precompiler processing is dependent on the use of the CLF declare commands FILE, INDS, and DCLFKSCLF. If a file is not declared within the CL program using FILE (or, if using CL syntax, DCLFCLF) then no additional processing of CLF commands, using the file, is done.

If no CLF declare commands are found by the precompiler then the precompiler uses the original source member when creating the module. This means that you can elect to always compile using the precompiler command.

The CRTCLFMOD command supports the same keyword parameters and values as the CRTCLMOD command. Many of the keyword values are passed through to the CRTCLMOD command as is. The major exception is related to the SRCFILE and SRCMBR keywords. Due to the precompiler generating a temporary source file member the values you specify for these two keywords are changed. The use of a temporary

source member also has an impact on the DBGVIEW keyword. The \*SOURCE option for DBGVIEW, though supported as a valid value, is not of practical value as the source member will most likely not exist at the time of attempting to debug the application module. A \*SOURCE view will only be available in the job that ran the precompiler and only so long as there is no additional use of the precompiler. For these reasons it is recommended that you specify DBGVIEW(\*LIST) if you anticipate needing to debug the application at a later time.

The CRTCLFMOD command can be used either interactively or in batch.

### ***Using the CRTCLFPGM Command***

The Create CLF Program (CRTCLFPGM) command runs the CLF precompiler. The precompiler processes your CL source prior to having the source statements compiled into a program. As a part of this processing the precompiler analyzes CLF commands, a subset of IBM provided commands, and any commands related to the definition of CL variables. A listing of the source examined and any errors that were detected is generated to help you correct any errors that occur.

The CLF precompiler creates a temporary source member that contains your original source and any expanded source that CLF has added. If no severe errors are found during the analysis phase the precompiler runs the appropriate IBM create command to compile this expanded source and create your application program. The object description for the created program object will reflect the name of your original source file and source member, not the name of the temporary source member the precompiler creates.

Precompiler processing is dependent on the use of the CLF declare commands FILE, INDS, and DCLFKSCLF. If a file is not declared within the CL program using FILE (or, if using CL syntax, DCLFCLF) then no additional processing of CLF commands, using the file, is done.

If no CLF declare commands are found by the precompiler then the precompiler uses the original source member when creating the program. This means that you can elect to always compile using the precompiler command.

The CRTCLFPGM command supports the same keyword parameters and values as the CRTCLPGM command. Many of the keyword values are passed through to the CRTCLPGM command as is. The major exception is related to the SRCFILE and SRCMBR keywords. Due to the precompiler generating a temporary source file member the values you specify for these two keywords are changed. The use of a temporary source member also has an impact on the OPTION keyword. The \*SRCDBG value for OPTION, though supported as a valid value, is not of practical value as the source member will most likely not exist at the time of attempting to debug the application program. A \*SRCDBG view will only be available in the job that ran the precompiler and only so long as there is no additional use of the precompiler. For these reasons it is

recommended that you specify `OPTION(*LSTDBG)` if you anticipate needing to debug the application at a later time.

The `CRTCLFPGM` command can be used either interactively or in batch.

## CONVENTIONAL THINKING CONCERNING FILE USAGE

If you are familiar with using files from conventional languages such as RPG or COBOL you can certainly continue to use the same thought processes when developing CLF-based applications that work with files. CLF provides the types of operations you would expect. Operations such as open a file, read a record, read the next or previous record, update a record, commit or rollback changes to the database, and close a file. All of these operations, and more, are available to you.

You are not however constrained to some of the limitations found with these other languages. For example conventional languages tend to require that you determine at compile time how you want to access a file. In RPG you decide at compile time if you want to read the record previous to the current record and you code a READP (Read Prior record) operation. Similarly in COBOL you might code READ PRIOR. With CLF you could code READRCDCLF TYPE(\*PRV) or the RPG-like READP command, but you could also code READRCDCLF TYPE(&DIRECTION) where the CL variable &DIRECTION is set to the value '\*PRV' (or a number of other values such as '\*NXT', '\*FIRST', '\*LAST', '\*SAME', etc). An example of using a CL variable for the TYPE keyword of the POSDBFCLF command can be found in the sample program DEV\_POSRD of source file VC2CLF/VC2CLSRC. The same program also demonstrates using a variable value for the Key Relation (KEYREL) keyword of the READRCDCLF command.

Most keywords of the CLF commands allow you to provide either a literal value, such as \*NXT, or a CL variable. Some keywords even support the use of CL built-ins such as %sst for substring operations. For some applications the flexibility of using CL variables and expressions may greatly simplify and reduce the amount of code you need to write and test.

Perhaps a bit extreme, but for 'thinking outside the box', you might consider that conventional file access typically requires that you declare what index (or key) is to be used in accessing data when you compile a program. RPG for instance determines index, or key characteristics, from your 'F' specification when you compile the program. But with CLF the index to be used does not have to be determined until you actually run the program.

The sample database file VC2EMP, provided with CLF, is defined with a number of character based fields. For demonstration purposes let's use a few of the logical files over some of these character fields. We will utilize two logical files: VC2EMPNAME and VC2EMPEXT. The DDS for these files is shown in Figure 4. 16 - DDS for logical file VC2EMPNAME and Figure 4. 17 - DDS for logical file VC2EMPEXT respectively.

```

.....A.....T.Name+++++.Len++TDpB.....Functions+++++
      R EMPRCD                               PFILE (VC2EMP)
      K EMPFNAME

```

Figure 4. 16 - DDS for logical file VC2EMPNAME

```

.....A.....T.Name+++++.Len++TDpB.....Functions+++++
      R EMPRCD                               PFILE (VC2EMP)
      K EMPEXT

```

Figure 4. 17 - DDS for logical file VC2EMPEXT

For a discussion of these files refer to Appendix B. VC2EMP and VC2DPT Sample Files.

Figure 4. 18 shows the source for sample program RPG\_MLTKEY. This program will use these two logical files though only one file is actually declared by the program.

```

Pgm      Parm(&Key)
Dcl      Var(&Key) Type(*Char) Len(32)

File     VC2EMP

Open     VC2EMP AccMth(*Key) LvlChk(*No)
Chain    &Key VC2EMP
SndPgmMsg Msg(&EmpFName *BCat ':' *Cat &EmpExt)

Close    VC2EMP
EndPgm

```

Figure 4. 18 - The RPG\_MLTKEY program

The RPG\_MLTKEY program is very simple. The program declares the file VC2EMP in order to extract the record layout of the file (this layout is the same as what is used for both of the logical files VC2EMPNAME and VC2EMPEXT), opens the file using a keyed access method, reads one employee record using the key value (&Key) passed in as a parameter to the RPG\_MLTKEY program, sends a message to the user containing

the first name and telephone extension of the employee found, closes the file, and exits the program.

The CL variable &Key is passed as a parameter to RPG\_MLTKEY and is defined as TYPE(\*CHAR) with a length of 32 bytes. The length of 32 bytes is chosen as that is the default length of a character string when calling a program from the command line. The actual key for the VC2EMPNAME logical file is 40 bytes, and for VC2EMPEXT 4 bytes. But as you will see shortly, this is not a problem due to the dynamic nature of CLF when working with the KEYLIST keyword. That is, assuming that you do not need to provide more than 32 bytes for an employee's first name when using the VC2EMPNAME file (in which case a command interface should be used).

There's no error checking in the program for conditions such as record-not-found but this can be added easily enough based on what you have learned earlier in this chapter (see Figure 4. 5).

The source for RPG\_MLTKEY can be found in member RPG\_MLTKEY of source file VC2CLF/VC2CLSRC. To create the sample program you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_MLTKEY) SRCFILE(VC2CLF/VC2CLSRC)
```

Prior to running RPG\_MLTKEY you should make sure that the VC2EMP physical file contains the expected records by reviewing Appendix B. VC2EMP and VC2DPT Sample Files. After this check, try a few different ways to call the RPG\_MLTKEY program.

From the command line run the commands:

```
OVRDBF FILE(VC2EMP) TOFILE(VC2EMPNAME)  
CALL PGM(RPG_MLTKEY) PARM(kiernan)
```

You should see the message 'Kiernan :1759' displayed. Even though the RPG\_MLTKEY program is compiled using the file VC2EMP, where the key is defined as a packed decimal value (EMPNBR of Figure B. 1), the RPG\_MLTKEY program has successfully performed a random read of the VC2EMPNAME logical file using a search key value of 'kiernan'. And due to the use of SRTSEQ(\*LANGIDSHR) when creating VC2EMPNAME, the search argument of 'kiernan' successfully returns the record with a key value of 'Kiernan'. Note though that the value returned to the application program is the actual, mixed case, value stored in the database.

Now run these commands:

```
DLTOVR FILE(VC2EMP)  
OVRDBF FILE(VC2EMP) TOFILE(VC2EMPEXT)  
CALL PGM(RPG_MLTKEY) PARM('8254')
```

You should see the message ‘Matthew :8254’ displayed. Without recompiling the RPG\_MLTKEY program you are now performing a random read of the VC2EMPEXT file using a character key value base on the telephone extension of the employee. Note that in this example, and the following one, you do need to put the key value you are searching for in quotes. This is due to the RPG\_MLTKEY program declaring the &KEY parameter as character (\*CHAR) while the system, if you were to call the program without the single quotes, would convert the numeric value 8254 to a packed decimal (\*DEC) format. The single quotes let the system know that this is a character value.

As one last experiment, run the following commands:

```
DLTOVR FILE(VC2EMP)
CALL PGM(RPG_MLTKEY) PARM('13')
```

You should see the message ‘Bruce :4178’. The RPG\_MLTKEY program, again without even a recompile, is now randomly accessing the VC2EMP index where the index is based on a packed decimal employee number field. The program is still successfully reading and displaying the employee information.

Do we need the OVRDBF commands used in the previous calls to RPG\_MLTKEY? As you might suspect, the answer is no. The FILE keyword of the OPEN command also accepts either a variable name or a literal. With a minor change to RPG\_MLTKEY you can eliminate the need for the override commands and essentially provide a generic search capability using the various logical file indexes.

Figure 4. 19 shows the source for one approach to eliminating the overrides.

```
Pgm          Parm(&File &Key)
Dcl        Var(&File) Type(*Char) Len(10)
Dcl          Var(&Key)  Type(*Char) Len(32)

File         VC2EMP

Open         VC2EMP File(&File) AccMth(*Key) LvlChk(*No)
Chain        &Key VC2EMP
SndPgmMsg   Msg(&EmpFName *BCat ':' *Cat &EmpExt)

Close       VC2EMP
EndPgm
```

Figure 4. 19 - The RPG\_MLTKEY2 program

The changes from the RPG\_MLTKEY sample program are shown in Figure 4. 19 using italicized bold. This version of the program is passed two parameters -- &FILE for the unqualified name of the file to use and &KEY. &KEY is used in the same manner as with RPG\_MLTKEY. The parameter &FILE is defined as TYPE(\*CHAR) with a length of 10

bytes. While the parameter will actually be passed as a 32 byte character field when called from the command line the program will only use the first 10 bytes. The &FILE parameter value is then used with the FILE keyword of the OPEN command to specify what file is to be actually opened at run-time.

The source for RPG\_MLTKEY2 can be found in member RPG\_MLTKEY2 of source file VC2CLF/VC2CLSRC. To create the sample program you can use one of the precompiler commands such as

```
CRTBNDCLF PGM(QTEMP/RPG_MLTKEY2) SRCFILE(VC2CLF/VC2CLSRC)
```

To run the program first make sure any file overrides from your testing of RPG\_MLTKEY are removed by running the command DLTOVR VC2EMP. Then run

```
CALL PGM(RPG_MLTKEY2) PARM(VC2EMPNAME KIERNAN)
CALL PGM(RPG_MLTKEY2) PARM(VC2EMPEXT '8254')
CALL PGM(RPG_MLTKEY2) PARM(VC2EMP '13')
```

The RPG\_MLTKEY and RPG\_MLTKEY2 examples demonstrate a capability that you will not use very often, if ever. They do however demonstrate that CLF is more than just eliminating the need for RPG, COBOL, or C database access programs when you need to work with files from CL applications. CLF opens up many opportunities for working with files on your Power i. RPG\_MLTKEY and RPG\_MLTKEY2 hopefully provide you with a feel for what you can do with CLF, and the data within your files, once you start ‘thinking outside the box’ of conventional file access. Conventional thinking based on RPG and COBOL file usage would, for many developers, cause them to never even consider using variables when opening a file (index) or when accessing a record within a file. CLF provides a level of flexibility that simply does not exist with languages such as RPG and COBOL – a flexibility that can significantly reduce the quantity of code (and programs) that you may need to develop and test. And this flexibility comes as an addition to the traditional file capabilities that you can continue to use.

## 5. Files, Record Formats, and Fields

CLF supports both externally-described and program-described files. This chapter provides information on how to use externally and program-described files within a CLF application.

This information is intended primarily as guidance to the CLF developer and represents a Programmers Guide level of information. CLF commands are discussed but not covered in detail. CLF command information at the Reference Manual level of detail can be found in the online help of CLF commands and also in HTML format at <http://www.brucevining.com>.

### USING EXTERNALLY DESCRIBED FILES

You can access the external definition of files using either the precompiler commands File Definition Specification (FILE) and Declare File using CLF (DCLFCLF). In addition, you can also use the run-time generation tool command Generate File Field Definition (GENFFDCLF).

In the following discussion any reference to the FILE command applies equally to the DCLFCLF command.

#### *Using the FILE and DCLFCLF command*

The FILE command is used in the source of a CL program or CL procedure to include record formats and field definitions into the CL source program being compiled by the precompiler. It is the combination of using the precompiler and declaring a file with the FILE command that enables much of the precompiler capability.

As with a standard DCLF command the FILE command must follow the PGM command and precede any commands other than other DCL, DCLF, DCLPRCOPT, DCLINDCLF, INDS, DCLFCLF, FILE, DCLFKSCLF, and COPYRIGHT commands.

The one required parameter of the FILE command is the file identifier (FILEID). This is the name used by all other CLF commands to reference the file.

The file (FILE) keyword is optional and identifies the actual name of the file object that is to be used when retrieving file, record format, and field level information. The FILE parameter defaults to using the FILEID name and locating the file using the library list of the job running the precompiler.

If a program requires that a given file be concurrently open more than once within the program there can be multiple FILE commands in the source program. The FILE commands can use the same FILE value so long as each FILE command has a unique FILEID specified.

When the precompiler encounters a FILE command in the CL source the FILE command is converted to a comment and DCLs equivalent to the record formats and fields defined by the referenced file are inserted into the CL source. By default all record formats defined by the file are declared. If only a subset of the record formats are needed by the application the RCDFMT keyword can be used to identify that subset.

In addition to creating DCL commands for the record formats and fields, the precompiler also extracts file-related information such as record format level check identifiers. This additional information is used by CLF run-time support to verify that the file being used at run-time is compatible with the file found at compile time. The default is to provide level checking when opening a file. This default behavior can be overridden by specifying LVLCHK(\*NO) when running the Open File for Processing (OPEN) command. Level checking is only performed for those record formats that were included in the compilation by the RCDFMT keyword of the associated FILE command.

The FILE Field Storage (FLDSTG) keyword has a significant impact on how the fields defined within the referenced file are declared within the CL application.

The default value of FLDSTG(\*AUTO) allows multiple definitions of a field to be accepted by the precompiler. This allows the application to declare multiple files using FILE where one or more of the files contain the same named field. The precompiler will declare the field only once and this single instance of the field will be shared across all files when commands such as CHAIN, WRITE, and UPDATE are run.

When using FLDSTG(\*AUTO) the first definition found for a field is the definition that is used when declaring the field. If, within the CL source program, the developer codes an explicit DCL for a field and later codes a FILE command that causes the precompiler to find another definition of the field, the precompiler will re-use the definition from the previous DCL. The ordering of DCLs, DCLFs, and FILE commands within the source program can then, when a field is defined with differing characteristics, be used to control which definition is used. If significant differences in the field definitions are detected the precompiler will log a message and stop the compile process.

An example of how field redefinition might be utilized is demonstrated in Figure 8. 14. The program shown is reading a file where field &EMPCOUNT, a count of employees in a department, is defined as a 2-digit packed decimal value. By preceding the FILE command for the file with the statement

```
DCL          VAR (&EMPCOUNT)  TYPE (*CHAR)  LEN (2)
```

The precompiler is able to map the numeric definition to a character definition. This character definition is then used as part of the MSG text with the SNDPGMMSG command. Without the precompiler the developer would normally need to define a separate variable declared as \*CHAR and use the CHGVAR command to convert the numeric value to the character value. This character based variable would then be used in the MSG text. By using the precompiler you can avoid having to declare a variable within the program, running the CHGVAR command, and coding the temporary variable as part of the message text.

If FLDSTG(\*DEFINED) is used the precompiler will declare multiple instances of a common field name, one per file (or record format for files containing multiple record formats) that contains the field. Unless the Name Modifier (NAMMDF) keyword is specified for all but one of the files (or all files, it's up to you) containing the common field name the CLF program will not create successfully. Similar to how the DCLF command requires a unique OPNID when multiple files are used within an application program in order to distinguish between common field names, CLF also requires a unique identifier. This unique identifier is specified using the NAMMDF keyword. Using FLDSTG(\*DEFINED) is quite similar to using RPG Input/Output commands to qualified data structure names.

FLDSTG(\*DEFINED) can also impact the names used for CL variables. For each field within a selected record format, one or more CL variables are declared by FILE. If a field is always at the same relative location within a record format one CL variable is declared. This CL variable will be named the same as the field in the record format after any user specified name modifications using the NAMMDF keyword are applied.

If a field is at different relative locations within a record format two CL variables are declared when using FLDSTG(\*DEFINED). Different relative field locations are typically found in \*DSPFs where input and output record format layouts are different due to the use of indicators and field usage specifications of input, output, and both to the fields within the record format. In this case CL variables that are input fields from the file will have the constant '\_I' appended to the end of the field name. Output fields to the file will have constant '\_O' appended to the end of the field name.

In addition to potentially changing field variable names, FLDSTG(\*DEFINED) may also modify record format names. If the record format name is the same as the file name then the constant '\_R' is appended to the end of the record format name. This name modification does not occur when FLDSTG(\*AUTO) is used with the FILE command.

Regardless of the FLDSTG value, if you have a field name within a file and the field name is the same as a FILEID name within the application then you must use the NAMMDF keyword with a value other than \*NONE for the file containing the field. If you do not use name modification the compile will fail due to duplicate name definitions. This situation typically occurs when using FILE with a file that was not created with

either DDS or SQL DDL. That is, the file was created as program described using the RCDLEN keyword of the CRTPF command.

## **Using the GENFFDCLF command**

Note that file/record/field definitions generated by the GENFFDCLF command cannot be used with RPG-like CLF commands. These definitions can only be used by CL syntax CLF commands.

The GENFFDCLF command generates DCLs describing a file, the record formats within the file, and the fields within each record format. The command is generally not used in the source of a CL program or CL procedure. GENFFDCLF is intended to allow a developer to include record formats and field definitions into a CL source program that is being compiled by the CRTBNDCL or CRTCLPGM commands and not a CLF precompiler. This command would typically be run interactively by a developer. If GENFFDCLF is coded in a CL source program it must follow all declare commands and global MONMSG monitors.

The CLF precompiler interprets DCLs generated by the GENFFDCLF command as normal DCL statements. The precompiler does not do any special processing of the file as it related to other CLF commands. Precompiler functions are only available when you declare a file using the FILE (or DCLFCLF) command.

The one required parameter of the GENFFDCLF command is the file identifier (FILEID). This is the name that all other CLF commands will use to reference the file.

The special value \*NONE can be used for the FILEID parameter if you want to generate CL source statement declares for the record and fields defined within a file (and not file level information). These source statements could then be copied or included into a non-CLF CL application program, into application documentation, etc. As there is no FILEID associated with these declares, CLF commands would not be able to use them implicitly. The record format and field declares can be used explicitly by way of the RCDBUF keyword available with many CLF commands.

The file (FILE) keyword is optional and identifies the actual name of the file object that is to be used when retrieving file, record format, and field level information. The FILE parameter defaults to using the FILEID name and locating the file using the library list of the job running the GENFFDCLF command. When FILEID(\*NONE) is specified you must provide a valid value for the FILE parameter. The default value of \*FILEID is not supported with FILEID(\*NONE).

In addition to creating DCL commands for the record formats and fields, file-related information such as record format level check identifiers can also be generated. This additional information is used by CLF run-time support to verify that the file being used at run-time is compatible with the file found when running the GENFFDCLF command. The default is to provide level checking when opening a file. This default behavior can be overridden by specifying LVLCHK(\*NO) when running the Open File for Processing

(OPEN) command. Level checking is only performed for those record formats that were included in the compilation by the RCDFMT keyword of the FILE command.

The GENFFDCLF command generates DCLs equivalent to the record formats and fields defined by the referenced file and writes these DCLs to the source file and member identified by the SRCFILE and SRCMBR keywords. By default all record formats defined by the file are declared. If only a subset of the record formats are needed by the application the RCDFMT keyword can be used to identify that subset.

For each record format selected, a CL variable representing the record buffer is declared by the GENFFDCLF command. This CL variable normally has the same name as the record format (after any user specified name modifications using the NAMMDF keyword are applied) and must be specified with the RCDBUF keyword of CLF Input/Output-related commands. The one exception to this naming convention is if the record format name is the same as the file name. In that case the constant ‘\_R’ is appended to the end of the record format name.

For each field within a selected record format, one or more CL variables are declared by GENFFDCLF. If a field is always at the same relative location within a record format one CL variable is declared. This CL variable will have the same name as the field in the record format after any user specified name modifications using the NAMMDF keyword are applied.

If a field is at different relative locations within a record format two CL variables are declared. Different relative field locations are typically found in \*DSPFs where input and output record format layouts are different due to the use of indicators and field usage specifications of input, output, and both to the fields within the record format. In this case CL variables that are input fields from the file will have the constant ‘\_I’ appended to the end of the field name. Output fields to the file will have constant ‘\_O’ appended to the end of the field name.

Unlike the FILE command GENFFDCLF does not allow the developer to select how field storage is to be managed. As the precompiler is not involved in creating the CLF application there is no opportunity to track field usage within the program and generate the necessary run-time instructions required to support FLDSTG(\*AUTO).

If you are creating an application that will be using the output of the GENFFDCLF command and the application has one or more of the following characteristics:

- Has explicit DCLs using the same name as the name of a field within a referenced FILE
- Uses one file where multiple record formats have fields of the same name

- Uses more than one file and another file has fields of the same name as the referenced FILE (this second file might be defined to the application using DCLF, FILE, DCLFCLF, or another GENFFDCLF)
- Uses a file where a field of the file is the same as the FILEID assigned to either the same file or another file

then unless the field name is changed to be unique the CLF program will not create successfully.

The one exception to this program creation failure is related to the multiple file case. If the GENFFDCLF generated DCL output is included into the CL source program prior to a FILE reference to the second file, and the FILE command is using FLDSTG(\*AUTO), then the program will still be able to create successfully when using the precompiler.

To change the generated field name the developer has two choices. One is to use the Name Modifier (NAMMDF) keyword of GENFFDCLF to modify the generated field name. The second choice is to edit the output of the GENFFDCLF command and change the name of the field prior to including the source into the application program. Once the DCLs have been generated the developer can edit the DCLs in order to meet the needs of the application program.

When editing the output of GENFFDCLF it is important that only the fields identified as being part of a record format be changed and that the *only* changes made are to the field name(s). Any other changes may result in run-time errors even though the CL program creates successfully.

## USING PROGRAM DESCRIBED FILES

Rather than using FILE, DCLFCLF, and/or GENFFDCLF commands, you can provide your own file definitions for one or more files within the CLF application. Note that program described files cannot be used with RPG-like CLF commands. These definitions can only be used by CL syntax CLF commands.

With program described files you must declare one or more CL variables of TYPE(\*CHAR). These variables will be used with the RCDBUF keyword of CLF Input/Output-related commands to identify the parameter that represents a record within the file. The application can, but does not need to, define subfields within the CL variable using the STG(\*DEFINED) and DEFVAR capabilities of the DCL command to represent fields of the record being worked with. These subfield definitions will not be used by CLF. They are only for your use. You can see examples of using STG(\*DEFINED) and DEFVAR by reviewing the output of the GENFFDCLF command.

Two examples of using program described files and base run-time support can be found in source file VC2CLF/VC2CLSRC. Member BAS\_OPNF demonstrates reading records from a source file without the use of STG(\*DEFINED) and DEFVAR. Member BAS\_OPNF1 demonstrates reading records from a source file using STG(\*DEFINED) and DEFVAR support. Both examples can be compiled using CRTBNDCL or CRTCLPGM.

If the application program will be reading records from the file the CL variable must be at least as large as the record that is to be read. If the application program will be updating records in the file the CL variable must be no larger than the record being updated. If the application program will be writing new records to the file the CL variable can be smaller, larger, or of the same length as the record format being written.

The Open File using CLF (OPNFCLF) command must specify LVLCHK(\*NO) when opening the program described file. Input/Output-related commands such as Read Record using CLF (READRCDCLF) and Update Record using CLF (UPDRCDCL) must have a TYPE(\*CHAR) CL variable coded for the RCDBUF keyword.

## USING EXTERNALLY AND PROGRAM DESCRIBED FILES IN THE SAME PROGRAM

You can use both externally-described and program-described files in the same CLF application. If you use the FILE command then you must use the precompiler to create the application, but the application program can also be using the output of the GENFFDCLF command and program-described files as part of the CL source code being compiled.

If you do combine file definitions in this manner the following considerations apply:

- The precompiler will only analyze those files that are declared using the FILE command.
- When using CL syntax CLF commands you can disable specific precompiler processing by explicitly specifying keywords that are normally provided by the precompiler. For example, the precompiler by default will determine the correct RCDBUF value to be used based on the RCDFMT (or FILEID if the file only defines one record format) you specify on a CLF Input/Output-related command. You can explicitly code the RCDBUF keyword, using a record buffer created by GENFFDCLF or yourself, and the precompiler will use the buffer you specify. Disabling the precompiler however should be avoided whenever possible. In the case of specifying your own RCDBUF value the precompiler will process the current CLF command as if the file had been declared with FLDSTG(\*DEFINED). Other CLF commands, where the RCDBUF keyword was not specified, would continue to operate with the FLDSTG value specified on the associated DCLFCLF command.

## USING CLF AND CL FILES IN THE SAME PROGRAM

You can use both CLF file capabilities and traditional CL file capabilities in the same CLF application. You can take an existing CL program that is perhaps using traditional commands such as DCLF and SNDRCVF to display information to a user and add additional files using CLF.

You may have an existing CL program that is:

- displaying information that is loaded into a \*DTAARA from a RPG program reading a database file, or perhaps the CL program is calling the RPG program and receiving the information to be shown by way of parameters
- allowing the user to change the information being displayed
- running CL commands based on the updates that the user entered in order to customize the flow of subsequent processing (perhaps printed output routing, file overrides, sending of messages to users, etc)

You can remove the \*DTAARA references (or program calls) and stop using (and maintaining) the RPG program. Whatever file function the RPG program was previously performing on behalf of the CL program can now be done in the CL program directly. You can have CL controlled files and CLF controlled files open and in use concurrently within your CL application program. The sample program RPG\_DSPU2 of Chapter 9: Using the Precompiler and CL File Support in the Same Application for instance demonstrates using a CL controlled \*DSPF with a CLF controlled database file.

Having both CLF and CL controlled files in the same application program provides you with greater flexibility than using only CL file support. With traditional CL support you must specify an OPNID for all files but one on the DCLF command. This OPNID then causes the field names associated with the declared file to be prefixed by the OPNID value. That in turn means you must use the CHGVAR command to move field values, originally with the same DDS or SQL names, from one file to another. Using CLF for the additional files allows you to maintain the original field names when using the precompiler.

When using both CL and CLF file support the following considerations exist:

1. the way that you open the file controls the set of commands that can be used to work with the file

- If the file is opened implicitly using CL commands such as RCVF and SNDRCVF then only IBM provided commands can be used with that instance of the open file
  - If the file is opened explicitly using CLF commands such as OPEN or OPNFCLF then only CLF provided commands can be used with that instance of the open file
2. within the same application you can open the same file twice using both traditional CL file support and CLF file support
- By default using the same file with both traditional CL commands and CLF provided commands will mean having two independent open instances of the file per item 1 above. That is, running the program shown in Figure 5. 1 will result in the employee name 'Rebecca' appearing twice. The RCVF command at (A) will read the first record of VC2EMP using the CL open instance. The READ command at (B) will read the first record of VC2EMP using the CLF open instance.

```

Pgm

DclF      File (VC2EMP)
File      FileID (VC2EMP)

Open      FileID (VC2EMP)

(A) RcvF
      SndPgmMsg  Msg (&EmpFName)

(B) Read      FileID (VC2EMP)
      SndPgmMsg  Msg (&EmpFName)

Close     VC2EMP

EndPgm

```

Figure 5. 1 - Independent opens of the VC2EMP database file

- By using the Override with Database File (OVRDBF) command and specifying SHARE(\*YES) you can have one shared open of the file. That is, running the program shown in Figure 5. 2 will result in the employee name 'Rebecca' being displayed first, followed by the employee name 'Bruce'. The RCVF command at (A) will read the first record of VC2EMP

using the shared open of VC2EMP. The READ command at (B) will read the next record of the VC2EMP shared open.

```
Pgm
DclF      File(VC2EMP)
File      FileID(VC2EMP)

OvrDBF    File(VC2EMP) Share(*Yes)

Open      FileID(VC2EMP)

(A) RcvF
      SndPgmMsg Msg(&EmpFName)

(B) Read      FileID(VC2EMP)
      SndPgmMsg Msg(&EmpFName)

Close     VC2EMP

EndPgm
```

Figure 5. 2 - Shared opens of the VC2EMP database file

3. In Figure 5. 2 you can use the DCLF and FILE commands in any order, and similarly for the RCVF and READ commands. In some situations, you may find that using the FILE command after the DCLF command provides for greater sharing of common field names.

The source for the program shown in Figure 5. 1, unique open instances with traditional CL file support, can be found in member RPG\_UNQCL of source file VC2CLF/VC2CLSRC. Assuming that library VC2CLF is in your library list (as that is where the VC2EMP database file can be found), RPG\_UNQCL can be created into QTEMP using

```
CRTBNDCLF PGM(QTEMP/RPG_UNQCL) SRCFILE(VC2CLF/VC2CLSRC)
```

The source for the program shown in Figure 5. 2, shared open with traditional CL file support, can be found in member RPG\_SHRCL of source file VC2CLF/VC2CLSRC. RPG\_SHRCL can be created into QTEMP using

```
CRTBNDCLF PGM(QTEMP/RPG_SHRCL) SRCFILE(VC2CLF/VC2CLSRC)
```

## FIELD INITIALIZATION CONSIDERATIONS

When a CL application program is called all CL variables declared in the program are initialized. This is standard behavior for CL programs and continues to be true for CLF applications. In a traditional CL program you can code the VALUE keyword on a DCL command and the CL variable will then be initialized to that value. If you do not code the VALUE keyword then traditional CL initialization will set character fields to blanks, numeric fields (decimal, integer, or unsigned integer) to 0, and logical fields to '0'.

In the past this initialization approach has worked reasonably well. But with the introduction of CLF you now have the ability to work with files that may contain date fields (DDS data type L), time fields (DDS data type T), timestamp fields (DDS data type Z), floating point fields (DDS data type F), 8-byte integer values (DDS data type B with a length of 18 digits), etc. From a CL point of view these are all character fields as CL does not provide direct support for any of these data types. Having CL initialize these fields to blanks (the default for character fields when you do not provide a DCL and associated VALUE) can introduce problems, especially when writing new records to a database file. To avoid these problems the CLF precompiler performs additional processing when working with fields.

When using the precompiler, and automatic field storage is specified with FILE FLDSTG(\*AUTO), the fields within the file are declared and initialized based on a variety of factors.

If you explicitly code a DCL for a CL variable and the variable name is the same as the name of a field defined within a file that is *later* processed by the FILE command, then your DCL is used. If your DCL uses the VALUE keyword, the value you specify will be the initial value of the CL variable. If you do not specify the VALUE keyword then CL default initialization will be used – blanks for TYPE(\*CHAR), 0 for TYPE(\*DEC), and '0' for TYPE(\*LGL).

If FILE processes a field within a file and no *previous* DCL has been encountered for the field name, then the precompiler will determine if a default value (DDS keyword DFT or SQL clause WITH DEFAULT) has been associated with the field. If so, the precompiler will initialize the field to the file (or table) defined default value.

If the field has not been defined with a default value then the precompiler will initialize the field based on the data type of the field. The initial values by data type are shown below.

File data type	CL program data type declared by CLF	Initial value
----------------	--------------------------------------	---------------

Fixed length character	*CHAR	Blanks
Varying length character	If VFLSPT(*VARLEN): 2-byte *INT *CHAR If VFLSPT(*FIXED) *CHAR	0 Blanks Blanks
Packed decimal	*DEC	0
Zoned decimal	*DEC	0
Binary numeric	*INT if 2 or 4 bytes in length *CHAR if greater than 4 bytes in length	0 Binary 0
Binary character	*CHAR	Initial byte of x'00' followed by blanks
Floating point	*CHAR	Floating point 0
Date	*CHAR	Blanks
Time	*CHAR	Blanks
Timestamp	*CHAR	Blanks
Hex	*CHAR	Blanks
Indicator	*LGL	Off

When you are not using the precompiler support, or if you specify FILE FLDSTG(\*DEFINED) for a file when using the precompiler, all record format subfields are initialized to blanks. This can cause data decimal errors in the case of packed decimal fields, incorrect value comparisons for logical fields, incorrect values for floating point fields, incorrect field lengths for variable-length fields, etc if the CL variable is used without first setting the CL variable to a valid (or correct) value. This initialization can be accomplished by you in many ways. Some methods include:

- using the CHGVAR command to set the CL variable to an appropriate value

- successfully reading a record where the record read contains a valid value (though note that a record-not-found condition will leave the CL variable at the original value)

## 6. Indicators

Many CLF commands utilize keyword parameters where the parameter is an indicator (or logical, \*LGL, CL variable). These indicator-based keywords are used when a condition can be either true or false. A true condition is when the CL variable has an 'on' (or '1') value. A false condition is when the CL variable has an 'off' (or '0') value. The use of indicators can simplify your coding efforts and better document within the CL source program your intent.

For example, the Read (READ) command defines the keyword EOF for End-Of-File. The CL variable you specify for the EOF keyword must be an indicator (logical variable). If in your program you code:

```
Read FileID(VC2DPT) EOF(&EOF)
```

Then the CL variable &EOF will be set to a true condition if end-of-file is reached on the read command. In your program you can test the &EOF condition in the following ways:

```
Read FileID(VC2DPT) EOF(&EOF)
If Cond(&EOF) Then(Do)
    /* Perform end-of-file processing */
EndDo
```

Alternatively you can test the &EOF condition with:

```
Read FileID(VC2DPT) EOF(&EOF)
DoWhile Cond(*Not &EOF) Then(Do)
    /* Perform record processing      */
    /* and then read the next record */
    Read FileID(VC2DPT) EOF(&EOF)
EndDo
/* Perform end-of-file processing */
```

For many developers either of the above coding styles is easier to follow than

```
Read FileID(VC2DPT) EOF(&EOF)
If Cond(&EOF *EQ '1') Then(Do)
    /* Perform end-of-file processing */
```

```

                                EndDo

Or

Read FileID(VC2DPT)
MonMsg MsgID(VC2501C) Exec (Do)
                                /* Perform end-of-file processing */
                                EndDo

```

As several CLF commands use indicator based parameters CLF provides commands to declare (DCL) indicators for you. You do not have to use the CLF provided indicator definitions. They are provided solely to simplify your coding efforts and to provide a set of naming standards that you may want to incorporate in your development efforts. The two precompiler based commands are Indicator Specification (INDS) and Declare Indicators using CLF (DCLINDCLF). The generation tool command is Generate Indicators using CLF (GENINDCLF).

## USING THE INDS COMMAND

The INDS command (also available as DCLINDCLF if using CLF CL syntax) is used in the source of a CL program or CL procedure to declare CLF-related indicators into the CL source program being compiled by the precompiler. As with other DCL-like commands the INDS command must follow the PGM command and precede any commands other than other DCL, FILE, DCLF, DCLPRCOPT, INDS, DCLINDCLF, DCLFCLF, DCLFKSCLF, and COPYRIGHT commands.

The INDS command supports the declaring of two distinct sets of indicators. You can declare one or both of these sets of indicators when you use the INDS command.

CLF input/output related indicators are declared when you specify CLFIND(\*YES). When this keyword is used the precompiler will declare the following three CL variables within your program:

CL Variable	Typical Keyword	Description
&EOF	EOF	End of file
&ERR	ERR	Error (such as record not available due to being locked by another job)
&RNF	RCDNOTFND	Record not found

These variables (&EOF, &ERR, and &RNF) would generally be used for the CLF command keywords EOF, ERR, and RcdNotFnd respectively. You can however use the CL variables in whatever manner best suits your needs. Likewise you can use INDS to declare all three indicators but elect to actually use none, one, two, or all three of the indicators with CLF commands. Or you could use an indicator with the EOF keyword for one READ command and omit the keyword on the next READ command in the CL program. It's all up to you.

The indicators are only updated when specified with a CLF command. There is no requirement to use the EOF, ERR, or RNF keywords. There is no requirement that you use the CLF declared indicators with the keywords. You can just as easily declare your own CL logical variable and use your variable.

When working with display (\*DSPF) files and printer (\*PRTF) files, some of the files might be using a separate indicator area (the DDS Indicator Area, INDARA, keyword) to isolate indicators from the record buffer associated with the file. The Indicator Area (INDARA) keyword of the INDS command will declare within your CL program a CL variable defining indicators &IN01 through &IN99. The name of the CL variable will be &CLF\_IND\_ with each individual indicator being defined as a subfield of the &CLF\_IND\_ variable. As with the CLFIND keyword of INDS you do not have to use this CLF provided indicator definition. You can define your own.

## **USING THE GENINDCLF COMMAND**

The GENINDCLF command is used to generate the same indicator definitions as are supported with the INDS command. The difference is that GENINDCLF support is provided by the CLF run-time generation tools option so there is no need to use a precompiler.

The source member created by GENINDCLF would then be copied (or INCLUDED) into your CL source program.

## **USING THE GENINDCLF COMMAND AND THE PRECOMPILER FOR THE SAME PROGRAM**

Though the precompiler supports the INDS command there are times you may find it useful to use both the precompiler and the GENINDCLF command. As the GENINDCLF command generates the indicator declares to a source file member you have the opportunity to customize the CLF assigned indicator names. If you have company conventions such as F3 on a display file always sets on indicator 03 and indicator 03 is always used to exit the application program, then you could go into the generated source

member and rename the generated field &IN03 to &Exit. Likewise &IN12 could, if appropriate, be renamed to &Return, &IN50 (if always used for record-not-found) might become &No\_Record, &IN99 (if always used for an error situation) might become &Error, etc. This customized source member could then be automatically imbedded into the CL program being compiled by a precompiler by use of the INCLUDE command. These renamed CL variables could also then be used with CLF command keywords such as ERR, EOF, and RcdNotFnd.

# 7. Detecting File Errors and Conditions

CLF categorizes file-related exceptions into two classes: error and condition.

An error generally represents a situation where corrective action must be taken before the program can run. Examples of an error would be attempting to open a file that does not exist or attempting to read from a file that has not been opened.

A file condition generally represents a situation that a program may encounter as part of normal processing. Examples of a file condition would be reaching end of file while reading a file sequentially or encountering a record not found condition when attempting to randomly read a record from a file.

CLF provides you with choices on how you want to detect both errors and conditions. You can monitor for escape messages, use CL logical program variables, or a combination of both approaches.

## MESSAGES

One approach to handling file-related exceptions, and one that most CL developers are quite familiar with, is monitoring for errors and conditions with the Monitor Message (MONMSG) command.

When working with error and file condition situations, CLF commands default to sending escape messages. These escape messages are documented in the online help for each command and are often preceded by one or more other messages that provide additional information on the error.

For instance, you might be attempting to open a file that does not currently exist. The Open File for Processing (OPEN) command documents that the escape message VC25017 (File ID &1 not open) can be sent. If we want to monitor for this error situation we can code:

```
Open          FileID(NOT_HERE)
MonMsg       MsgID(VC25017) Exec(Do)
              SndPgmMsg  Msg('My open failed')
              Return
              EndDo
```

If the file open fails the program shown will send the message 'My open failed' and then return. In the job log additional information will be found. In the case of the file not being found you will find CPF4101 - File &2 in library &3 not found.

In a similar fashion, file-related conditions such as record not found can be monitored by the application program. The Random Retrieval from a File (CHAIN) command documentation shows that VC2501B (Record not found in file &1) can be sent. The following code fragment is attempting to randomly read a record from the VC2EMP sample database where the employee number is 50.

```
Open          FileID(VC2EMP) AccMth(*Key)
Chain         50 VC2EMP
MonMsg       MsgID(VC2501B) Exec(SndPgmMsg Msg('Employee +
              not found.'))
```

If no record exists for employee number 50 the program will send the message 'Employee not found.' In this situation there will be no additional diagnostic messages found in the job log. The error message VC2501B sufficiently explains the situation.

Some CLF commands document that several possible escape messages may be sent. Though only message one will be sent during the running of a specific command, the command may send different messages depending on the environment the command is running in. In the example using the CHAIN command the application monitored for message VC2501B. If however we change the Open File for Processing (OPEN) command and remove the ACCMTH(\*KEY) parameter then the CHAIN command will send the message VC25021 (The READRCDCLF command was not successful when accessing File ID &1). This error message is sent because TYPE(\*KEY) is implied with the CHAIN command as the RRN keyword was not specified. The OPEN, on the other hand, is now defaulting with ACCMTH(\*RRN). To monitor for both situations you can:

```
Open          FileID(VC2EMP)
Chain         50 VC2EMP)
MonMsg       MsgID(VC2501B) Exec(SndPgmMsg Msg('Employee +
              not found.'))
MonMsg       MsgID(VC25021) Exec(SndPgmMsg Msg('Employee +
              file access problem.'))
```

Now when calling the program, after changing ACCMTH(\*KEY) to ACCMTH(\*RRN), the escape message VC25021 is sent and in your job log will also be VC25019 - File ID VC2EMP is not open for keyed access.

There can be up to 100 MONMSG commands following a CLF command and up to 1000 MONMSG commands within a program.

## Obtaining additional information from CLF messages

Several of the CLF error-related escape messages that you can monitor for provide additional information related to the underlying cause of the error. These messages are:

- VC25017 - File ID &2 not opened.
- VC2501E - Write to file ID &2 not successful.
- VC25021 - The &4 command was not successful when accessing File ID &2.
- VC25037 - Update operation to file ID &2 not successful.
- VC25038 - Delete operation to file ID &2 not successful.
- VC2503A - Write/Read to file ID &2 not successful.

Each of these messages provides a replacement text variable which identifies the most likely Power i message ID that more fully describes the error. This message ID is always the first replacement variable defined for the VC2 error messages shown above. When your application program is sent one of these VC2 generated messages, and you are monitoring for the error, you can receive the CLF message and then examine the message ID value.

### Example using the OPEN command

In the previous example of a file not being found and message CPF4101 being in your job log, you could have coded the application as shown below.

```
Dcl          Var(&MsgDta) Type(*Char) Len(7)

Open        FileID(NOT_HERE) LvlChk(*No)
MonMsg      MsgID(VC25017) Exec(Do)
            RcvMsg MsgType(*Last) MsgDta(&MsgDta)
            If Cond(&MsgDta = CPF4101) +
              Then(SndPgmMsg Msg(+
                  'The file was not found.'))
            Else SndPgmMsg Msg( +
                  'Error' *BCat &MsgDta +
                  *BCat 'encountered.')
```

Return  
EndDo

Assuming that you do not have a file named NOT\_HERE in your library list, the above program will send the message 'The file was not found' when run. The program, in addition to the MONMSG for VC25017 that you saw previously, is now receiving the VC25017 message as part of the MONMSG processing. This is done with the Receive Message (RCVMSG) command. When receiving the message, the program is also requesting that the first seven bytes of message replacement data be returned in the CL variable &MsgDta. This is done with the MSGDTA keyword of the RCVMSG command. The VC25017 message actually provides more replacement data than just seven bytes, but as all you are concerned with (at this point anyway) is the message ID associated with the system error message you only need to declare the &MSGDTA CL variable as being seven bytes in length. To see what additional information is available to your application program you can use the Display Message Description (DSPMSGD) command as shown here

```
DSPMSGD RANGE(VC25017) MSGF(VC2CLF/VC2MSG)
```

Using the DSPMSGD command you can see how replacement data variables are used and declared within the message. In the case of the message ID variable, it is used as replacement variable &1 and defined as a \*CHAR data type of length 7.

Having received the message the program compares the received &MSGDTA value to the constant CPF4101. If equal the message 'The file was not found' is sent to the user of the program. If the comparison is not equal, a message that includes the system error message ID is sent to the user of the program. Your own application program could of course do much more than shown. For instance if &MSGDTA is equal to CPF4101 it might be appropriate to create the file, re-run the OPEN command, and then continue processing.

#### Example using the WRITE command

The following program is a version of the RPG\_ADDUSR CPP from Chapter 4: Writing a Record to a Database File. The original program attempted to read the PRTCTL record for a user/report combination prior to writing the record. This was to avoid duplicate key errors when running the program. This version does not attempt to read a record from the PRTCTL database prior to writing a record. This example will determine if a failure with the WRITE command is due to a duplicate key, the file being at maximum capacity, or some other cause.

```
/* ***** */
/* This program is the CPP for the ADDPRTUSR */
/* command. */
/* ***** */
```

```

Pgm          Parm(&UsrPrf &PrtF &OutQ &Copies)

/*****
/* No need to declare the parameters as they */
/* have the same name as the fields within the */
/* PRTCTL database file. Do need to declare */
/* &MsgID so we can get error details. */
*****/

Dcl          Var(&MsgID) Type(*Char) Len(7)

/*****
/* Declare the PRTCTL file */
*****/

File          FileID(PRTCTL)

/*****
/* Open PRTCTL for output and write the record */
/* Monitor in case an error is encountered */
*****/

Open          PRTCTL Usage(*Output)

Write         PRTCTL
MonMsg        MsgID(VC2501E) Exec(Do)
              RcvMsg MsgType(*Last) MsgDta(&MsgID)
              Select
                When Cond(&MsgID = CPF5026) Then( +
                  SndPgmMsg Msg('Report' *BCat +
                    &PrtF *BCat 'is currently +
                    defined for user' *BCat +
                    &UsrPrf *TCat '. Use the +
                    CHGPRTUSR command to +
                    change the existing entry.))
                When Cond(&MsgID = CPF5018) Then(Do)
                  SndPgmMsg ToUsr(*SYSOPR) +
                    Msg( +
                      'File PRTCTL is at capacity. +
                      Please schedule the necessary +
                      changes. ')
                  SndPgmMsg Msg('Unable to add new +
                    users to PRTCTL. The system +
                    operator has been notified. +
                    Try again later. ')
                EndDo
                Otherwise Cmd(Do)
                  SndPgmMsg ToUsr(*SYSOPR) Msg( +
                    'Unexpected failure' *BCat +
                    &MsgID *BCat 'encountered by +
                    RPG_ADDUS1. Please notify +
                    development immediately. ')

```

```

                SndPgmMsg Msg('Unable to add new +
                users to PRTCTL. The system +
                operator has been notified. +
                Try again later.')
            EndDo
        EndSelect
    Close PRTCTL
    Return
EndDo

SndPgmMsg Msg('Report' *BCat &PrtF *BCat +
'successfully added for user' *BCat +
&UsrPrf *TCat '.')

Close    PRTCTL

Return
EndPgm

```

Figure 7. 1 - Determining specific error causes with WRITE

If you compare the source of Figure 7. 1 with the source of Figure 4. 5 you will find that there are a few differences. Rather than attempting to read a record from PRTCTL, using the specified &USRPRF and &PRTF parameter values, the program now simply writes a new record to PRTCTL using the WRITE command and, if no error is returned, sends a message to the user indicating that the record was added, closes the file, and returns.

But if an error is encountered there is quite a bit more application logic involved. As reviewed in the previous topic, Example using the OPEN command, you now use the RCVMSG command to receive the error message sent by the WRITE command along with having the CL variable &MSGID. After running the RCVMSG command the CL variable &MSGID will be set to the value of the system error message most likely describing the error. Following this the program examines the value of &MSGID and takes corrective action on the error when possible.

The first check, if &MSGID is set to CPF5026 (Duplicate key not allowed for member &4), is handled in the same manner as in RPG\_ADDUSR -- the program notifies the user that the report for this user is already in the PRTCTL database and to use the CHGPRTUSR command if they need to change the entry.

The second check however is new. Here the program determines if the &MSGID is CPF5018 (Member &4 at maximum size. Increment not allowed) and, if so, sends a message to the system operator reporting the problem and asking that it be corrected. The program in this situation also sends a message to the user letting them know that a problem has been found, that the system operator has been informed of the problem, and that they should try again later.

If &MSGID is not either CPF5026 or CPF5018 the program takes yet another recovery action. In this case, where the program has not been written to handle the specific error, the system operator is again sent a message. But this time the message reports the problem and asks the operator to inform development that an unexpected error was encountered along with the error message that was received. The program also then sends a message to the user letting them know that a problem has been found, that the system operator has been informed of the problem, and that they should try again later.

The source for this program can be found in member RPG\_ADDUS1 of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use

```
CRTBNDCLF PGM(QTEMP/RPG_ADDUS1) SRCFILE(VC2CLF/VC2CLSRC)
```

To change the ADDPRTUSR command to use QTEMP/RPG\_ADDUS1 as the CPP you can use

```
CHGCMD CMD(ADDPRTUSR) PGM(QTEMP/RPG_ADDUS1)
```

## PROGRAM VARIABLES

As an alternative to escape messages CLF provides keyword parameters where you can provide CL logical variables to have error and condition information status returned. From a system processing point of view using program variables is more efficient than sending, and monitoring for, escape messages.

Most CLF commands provide an error keyword (ERR) that can be used to detect if an error was encountered during the running of the command. The CL variable must be defined as TYPE(\*LGL). If after running the command the variable is set to a value of '1' (on) then an error was detected. If the variable is set to a value of '0' (off) then no error was found in the running of the command.

You can change from using escape messages to using CL program variables by specifying the appropriate keywords as shown below.

```
Dcl          Var(&Err_Found) Type(*Lgl)

Open        FileD(VC2EMP) ERR(&Err_Found)
If          Cond(&Err_Found) Then(Do)
           SndPgmMsg Msg('My open failed')
           Return
           EndDo
```

In the previous example the program declares the logical variable &ERR\_FOUND. This variable is then used with the ERR keyword of the OPEN command. If the &ERR\_FOUND variable is 'on' after running the OPEN command an error was found by the command. The message 'My open failed' is then displayed by the application program. As with escape messages, if the file does not exist the message CPF4101 will be found in the job log to assist in problem determination.

Note that if you use a keyword such as ERR then no VC2 escape message will be sent for you to monitor. Your error detection, for a given condition with a specific CLF command, is either program variable based or escape message based.

### ***Having CLF declare your program variables for you***

Rather than coding your own DCLs for these indicators or logical variables, CLF provides commands to declare a standard set of CL variables to associate with the ERR, RCDNOTFND, and EOF keywords. These variables are &ERR, &RNF, and &EOF respectively. These variables, when using the precompiler, can be declared using the Indicator Specification (INDS) command with CLFIND(\*YES). When not using the precompiler you can use the Generate Indicators using CLF (GENINDCLF) command with CLFIND(\*YES).

If you were using CLF created logical variables with the precompiler, then the previous example would become:

```

Inds          CLFInd(*Yes)

OpnFCLF      FileID(VC2EMP) AccMth(*Key)
Chain        50 VC2EMP RcdNotFnd(&RNF) Err(&ERR)
If           Cond(&RNF) Then(SndPgmMsg Msg( +
                'Employee not found.'))
If           Cond(&ERR) Then(SndPgmMsg Msg( +
                'Employee file access problem'))

```

When using either INDS CLFIND(\*YES) or GENINDCLF CLFIND(\*YES) the following declares are created for your use.

```

DCL          VAR(&EOF)    TYPE(*LGL)  /* End of file      */
DCL          VAR(&ERR)    TYPE(*LGL)  /* Error found      */
DCL          VAR(&RNF)    TYPE(*LGL)  /* Record not found */

```

## **MESSAGES AND PROGRAM VARIABLES**

It is up to you as the developer to decide whether you want to work with escape messages, program variables, or both.

You can elect to use escape messages with one instance of a command within an application program and program variables with another instance of the command later in the program. The one consideration is that you will not receive an escape message when the equivalent command keyword is also being used.

For example the MONMSG statement shown below will never run. The READ command is coded using the EOF keyword and the equivalent escape message for end of file (VC2501C) will never be sent.

```
Read          FileID(VC2EMPDPT) EOF(&EOF) +  
MonMsg       MsgID(VC2501C) Exec(Leave)
```

## 8. Using Database Files

You can access a database file from your program by opening the file with the Open File for Processing (OPEN) command or, if using CL syntax, the equivalent Open File using CLF (OPNFCLF) command. On the OPEN command you provide a File identifier (FILEID) that is used by other CLF commands when referencing the file. The FILE keyword of the OPEN command allows you to optionally specify the database file and library to associate with the FILEID. The database file, when initially opened, is positioned prior to the first record of the file.

The database file you open can be either physical or logical and either externally described or program-described. The files can also be on remote systems when using the distributed data management (DDM) capabilities of the i operating system. Externally described file descriptions are accessed using the File Description Specification (FILE) command, the Declare File using CLF (DCLFCLF) command, or the Generate File Field Definitions (GENFFDCLF) command. An introduction to these commands can be found in Chapter 5. Files, Record Formats, and Fields.

The RPG-like commands that are supported for database files are:

- CHAIN – Random read of a record
- CLOSE – Close a file
- DELETE – Delete a record from a file
- OPEN – Open a file
- READ – Read the next record within a file
- READE – Read the next record within a file if key is equal to a specific value
- READP – Read the previous record within a file
- READPE – Read the previous record within a file if key is equal to a specific value
- SETGT – Position a file to a location greater than a specific value
- SETLL – Position a file to a location equal to or greater than a specific value
- UNLOCK – Release the lock on a record

- UPDATE – Update a record within a file
- WRITE – Write a new record to a file

In addition the following CL-syntax CLF commands can be used with database files.

- CLOFCLF – Close a file
- DLTRCDCLF – Delete a record from a file
- FRCDTACL – Force data to permanent storage
- OPNFCLF – Open a file
- POSDBFCLF – Position a file to a given location
- READRCDCLF – Read a record from a file
- RLSRCDCLF – Release the lock on a record
- RTVFINFCLF – Retrieve information about a file
- UPDRCDCLF – Update a record within a file
- WRKOPNFCLF – Work with open files
- WRTRCDCLF – Write a new record to a file
- WRTRRNCLF – Write a new record to a file by relative record number

The following IBM provided commands can be used with CLF database files:

- COMMIT – Commit database transactions
- OVRDBF – Override file used or file parameters
- ROLLBACK – Rollback database transactions

The following CLF precompiler commands can be used with fields within a database record:

- CHGNULACLF – Change the null attribute associated with a field
- RTVNULACLF – Retrieve the null attribute associated with a field

- RTVVFLCLF – Retrieve the length of a variable-length field
- SETVFLCLF – Set the length of a variable-length field

The following CLF precompiler commands can be used to extract file, record format, field, and key information from a database file:

- FILE – Extract file, record format, and field information during the precompile step of program creation
- DCLFKSCLF – Extract key field information during the precompile step of program creation
- DCLFCLF – Extract file, record format, and field information during the precompile step of program creation

The following CLF generation tool commands can be used to extract file, record format, field, and key information from a database file:

- GENFFDCLF -- Generate file, record format, and field information to be stored in a source file member
- GENFKSCLF – Generate key field information to be stored in a source file member

Refer to the online help text provided with all CLF commands for information related to a given command's usage and considerations.

Most of the RPG-like CLF database commands have a minimal number of parameters and are easily understood. Commands such as Create New Records (WRITE), Modify Existing Record (UPDATE), and Delete Record (DELETE) do not require a large amount of explanation in order for you to understand how to use them. The help text associated with each of these commands, and the examples provided in the help text, should have you productively using these functions very quickly. You will however find examples of using these commands sprinkled throughout this manual. One simple example can be found at Example using Database Read, Delete, and Update with the CLF Precompiler later in this chapter.

One facet of database use does however warrant an extended discussion. That is the use of key values when reading or positioning to a record within the database. Prior to a discussion of key values though will be an introduction to access paths – the ways by which you can access your database information.

## ACCESS PATHS

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (non-keyed) access path or on a keyed-sequence access path. The arrival sequence access path is based on the order the records are stored in the file. Records are generally added to the file one after another. For the keyed-sequence access path, the sequence of records in the file is based on the contents of the key fields defined for the file. CLF supports both arrival sequence and keyed-sequence access paths. You select the access method to use with the ACCMTH keyword of the OPEN command. Within the selected access path, CLF supports both sequential and random processing of records. With sequential processing records are read in the order they appear in the access path. With random processing records are read by either a key value or a relative record number value. You can use both random and sequential processing for the same file. For example you can use the CHAIN command to randomly read a record by key and then read the next sequential record (READ) from your current position in the file.

The following examples demonstrate how the access method (ACCMTH) keyword controls the order of records read from a file. While these example programs are complete in terms of showing how ACCMTH can change the behavior of your application programs, they are not complete in terms of having any form of error handling. Actual applications should, at a minimum, be handling record-not-found conditions.

The file being used is the VC2EMP employee master database file that is provided with CLF. Before running the sample programs you should verify that this file contains the expected records. You can do this with the instruction provided in Appendix B. VC2EMP and VC2DPT Sample Files.

Figure 8. 1 shows how using an access method based on arrival sequence controls how records are processed by an application program. Figure 8. 2 shows how using an access method based on key-sequence changes how the same records are processed.

Pgm	Parm (&Value)
Dcl	Var (&Value) Type (*Dec)
File	VC2EMP
<b>(A)</b> Open	VC2EMP <b>AccMth (*RRN)</b>
<b>(B)</b> Chain	FileID (VC2EMP) <b>RRN (&amp;Value)</b>
SndPgmMsg	Msg (&EmpFName)
<b>(C)</b> Read	VC2EMP

```

SndPgmMsg  Msg (&EmpFName)

Close      VC2EMP
EndPgm

```

Figure 8. 1 - RRN access by arrival sequence

The program in Figure 8.1 takes one parameter, &VALUE. &VALUE is declared as TYPE(\*DEC) and allowed to default to LEN(15 5). This default matches what the system will pass if, as you will do shortly, you call this program from a command line. The program also declares the file VC2EMP using the FILE command.

At (A) the VC2EMP file is opened using the OPEN command with ACCMTH(\*RRN) specified. This indicates to CLF that your access to the records contained within the VC2EMP database file will be based on arrival sequence. You do not have to actually specify ACCMTH(\*RRN) as this is the default access method, but in the example it was thought best to explicitly show the keyword and value.

At (B) the program uses the CHAIN command to randomly read a record from the VC2EMP database file. The program then sends a message containing the first name of the employee record found. The record being randomly read is the record found at the relative record location, based on the arrival sequence access path of VC2EMP, identified by the value of the &VALUE CL variable. This is specified with the RRN(&VALUE) keyword of the CHAIN command.

At (C) of Figure 8.1 the program sequentially reads the next record from the VC2EMP database file using the READ command and sends a message containing the first name of the employee record found. The read is based on arrival sequence due to ACCMTH(\*RRN) attribute on the OPEN command.

The program then closes the VC2EMP file and returns.

The source for Figure 8.1 is in member RPG\_ACCRRN of source file VC2CLF/VC2CLSRC. To compile this program into QTEMP use the commands

```

ADDLIBLE LIB (VC2CLF)
CRTBNDCLF PGM(QTEMP/RPG_ACCRRN) SRCFILE (VC2CLSRC)

```

Now call the program and see what the program returns. First call RPG\_ACCRRN using

```
CALL PGM(QTEMP/RPG_ACCRRN) PARM(13)
```

You should see displayed the name 'Wayne' followed by the name 'Clayton'. If you run the command

```
DSPPFM FILE (VC2EMP)
```

you will find that the thirteenth record in the file is 'Wayne' and the fourteenth record 'Clayton'.

Figure 8. 2 shows a program where the only change from Figure 8. 1 is the ACCMTH keyword of the OPEN command. This change is shown at (A) of Figure 8. 2.

```
Pgm      Parm(&Value)
Dcl      Var(&Value)  Type(*Dec)

File     VC2EMP

(A) Open  VC2EMP  AccMth (*Key)

Chain    FileID(VC2EMP)  RRN(&Value)
SndPgmMsg  Msg(&EmpFName)

Read     VC2EMP
SndPgmMsg  Msg(&EmpFName)

Close    VC2EMP
EndPgm
```

Figure 8. 2 - RRN access by keyed sequence

This small change will have a large impact on the order in which records are read from the VC2EMP database file.

The source for Figure 8. 2 can be found in member RPG\_ACCKR of source file VC2CLF/VC2CLSRC. To create this program into QTEMP use the command

```
CRTBNDCLF PGM(QTEMP/RPG_ACCKR) SRCFILE(VC2CLSRC)
```

Now call the program with the command

```
CALL PGM(QTEMP/RPG_ACCKR) PARM(13)
```

You will see that the first name shown continues to be 'Wayne'. This is because random using RRN is always based on the arrival sequence access path. But the second name shown is now 'Richard', as opposed to 'Clayton' when running the RPG\_ACCRRN program.

The reason for this difference is based solely on the ACCMTH selected when opening the VC2EMP file. With running the program RPG\_ACCRRN sequential access is based on the arrival sequence access path of the file. When running RPG\_ACCKR sequential access to the file is based on the keyed access path. For VC2EMP the keyed access path is based on the employee number (&EMPNBR). 'Wayne' is employee number 53 and 'Richard', employee number 107, has the next higher employee number than 'Wayne'.

Figure 8. 3 shows one more variation on this topic.

```
Pgm          Parm(&Value)
Dcl          Var(&Value) Type(*Dec)

File         VC2EMP

Open         VC2EMP AccMth(*Key)

(A) Chain    FileID(VC2EMP) KeyList(&Value)
SndPgmMsg   Msg(&EmpFName)

Read        VC2EMP
SndPgmMsg   Msg(&EmpFName)

Close       VC2EMP
EndPgm
```

Figure 8. 3 - Keyed access by key sequence

The change in Figure 8. 3 from Figure 8. 2 is shown at (A) of Figure 8. 3. The program has been changed to use the KEYLIST keyword rather than the RRN keyword when randomly accessing the database file. The key value you want to use continues to be identified by the &VALUE parameter.

The source for Figure 8. 3 can be found in member RPG\_ACCKEY of source file VC2CLF/VC2CLSRC. To create this program into QTEMP you use the command

```
CRTBNDCLF PGM(QTEMP/RPG_ACCKEY) SRCFILE(VC2CLSRC)
```

To run the program use

```
CALL PGM(QTEMP/RPG_ACCKEY) PARM(13)
```

You will now see that the names displayed are 'Bruce' and 'Mindy' rather than 'Wayne' and 'Richard'. 'Wayne' has been replaced in the displayed list as RPG\_ACCKEY is now performing a random read based on the keyed access path, and 'Bruce' is employee number 13. This change is due to the use of KEYLIST with the CHAIN command.

'Mindy' is now shown, rather than 'Clayton' or 'Richard', as the second name because 'Mindy' is employee number 40. Employee number 40 is the next higher employee number after employee number 13.

As you can see, relatively minor changes in ACCMTH and RRN/KEYLIST can make significant changes in the order by which records are accessed by your application.

## KEY VALUES

For keyed-sequence access paths you can define one or more fields as being key fields for a record format. CLF supports full key and partial key access when working with a file defined with multiple key fields. In order to use keys when working with a database you must specify ACCMTH(\*KEY) when opening the file with the OPEN command.

In your CLF program you specify a search argument on the keyed read command (CHAIN, READE, or READPE) or keyed positioning command (SETLL or SETGT) to identify the record you want to process. The search argument corresponds to the key values you are interested in.

When reading from a file using a keyed-sequence access path there are two ways you can specify key values. They are the keywords KEYLIST and KEYSTRUCT. When positioning to a record using a keyed-sequence access path there are three ways you can specify key values. They are the keywords KEYLIST, KEYSTRUCT, and KEYCOUNT.

KEYLIST allows you to specify up to 10 CL variables within a list. Each variable corresponds to one key field in the file's key definition. This support is very flexible in that the definition of the CL variable does not have to match the definition of the key field within the file. The two definitions can be of different TYPE and LEN. The KEYLIST parameter also does not need to have a key value for each key field defined within the file. If a file is defined with three key fields, you can specify CL variables for one, two, or all three of the key fields. You cannot however 'skip' a key value. If you want to specify a value for the second key field you must also specify a value for the first key field. One consideration when using the KEYLIST parameter with numeric key values is that the value must be within the range of .00001 to 9999999999.99999 (positive or negative). If your key values are larger, or smaller, than this range then you must use the KEYSTRUCT or KEYCOUNT parameter.

KEYSTRUCT allows you to specify the number of key values to be used and a structure that contains the values. This structure can be created for you by using either the precompiler Declare File Key Structure (DCLFKSCLF) command or the generation tool Generate File Key Structure (GENFKSCLF) command. KEYSTRUCT allows you to easily change the number of key values used when positioning or reading in a file as the number of key fields element of the parameter can be a CL variable. KEYSTRUCT allows you to specify numeric key values outside of the range of .00001 to 9999999999.99999 (positive or negative). KEYSTRUCT is similar to the support found with RPG %KDS built-in.

KEYCOUNT does not allow you to explicitly specify the key values to be used. KEYCOUNT allows you to specify *the number of key field values*, from the record you are currently positioned at, to use in order to access the requested record. The values

come from the current record, and not the variable values within your application program.

## KEY RELATION (KEYREL) EXAMPLES

The following examples, which show how various keyed operators work, use the employee master logical file VC2CLF/VC2EMPNAME. VC2EMPNAME is based on the VC2EMP physical file and defined with a key based on the employee first name. The key uses a shared value sort sequence table (see Appendix B. VC2EMP and VC2DPT Sample Files if you need a review of this logical file). VC2EMP should contain 15 employee records. When accessed using the VC2EMPNAME logical file the full list of employees are sequenced in this order:

1. Amie
2. Betty
3. Bruce
4. Clayton
5. Dovid
6. Hadleigh
7. Kiernan
8. Matthew
9. Mindy
10. Moshe
11. Rebecca
12. Richard
13. Richard
14. Ruth
15. Wayne

The various read commands will be discussed first, followed by the positioning commands. But before getting into the examples, if you have not previously done so, verify the contents of the VC2EMP file by using the instructions found in Appendix B. VC2EMP and VC2DPT Sample Files.

### ***Read examples***

#### **Random access**

When reading records by key you can access the records either randomly or relative to your current location in the file. This section reviews the random access by key functions available to you. RPG provides the CHAIN operation code which provides for random access for an \*EQ key and is implemented in CLF with the CHAIN command. CHAIN is quite straightforward, you provide the key values using either the KEYLIST or KEYSTRUCT keyword and the record with a matching key, if found, is returned. CLF however provides for many more forms of random access if you step out of the RPG-like commands. The CL-syntax READRCDCLF command for instance provides for key

comparisons of less than or equal to (\*LE), greater than or equal to (\*GE), less than (\*LT), and greater than (\*GT). As you can freely intermix RPG-like and CL-syntax commands when working with a file this section will look at some of the READRCDCLF functions available to you.

To demonstrate how the various key comparisons work one command and one program will be written. The command will prompt for a valid key relation value (\*EQ, \*GT, etc) and the first name of an employee. The program will be the CPP for the command and define two parameters – &KEYREL and &NAME. These parameters will then be used to specify the KEYREL value you want to test and the employee first name key value you want to use. Figure 8. 4 shows the source for the command Random Access (RNDACS). Figure 8. 5 the source for the program.

```

CMD          PROMPT('Random Access KEYRELS')
PARM        KWD(KEYREL) TYPE(*CHAR) LEN(10) RSTD(*YES) +
            VALUES(*EQ *LE *GE *LT *GT) MIN(1) +
            PROMPT('Key relation test')
PARM        KWD(NAME) TYPE(*CHAR) LEN(40) MIN(1) +
            CASE(*MIXED) PROMPT('Employee first name')

```

Figure 8. 4 - The RNDACS command source

The RNDACS command defines two required parameters – KEYREL and NAME. KEYREL represents the key relation that you want to use and supports the random access values \*EQ, \*LE, \*GE, \*LT, and \*GT. NAME represents an employee first name and can be up to 40 bytes in length. The NAME parameter will not be converted to uppercase even if you don't use single quotes around the name. While the sample program uses the READRCDCLF command you could implement any of the \*EQ KEYREL values with the CHAIN command.

The source for the RNDACS command can be found in member RNDACS of the source file VC2CLF/QCMDSRC. To create the RNDACS command into QTEMP you use the command

```

CRTCMD CMD(QTEMP/RNDACS) PGM(QTEMP/RPG_RNDACS) +
        SRCFILE(VC2CLF/QCMDSRC)

```

```

/*****
/* Define two parameters:
/*   &KeyRel - The Key Relation (KEYREL) that is to be used
/*   &Name   - The employee name to search with in
/*             conjunction with the &KeyRel parameter
*****/

Pgm          Parm(&KeyRel &Name)

```

```

Dcl          Var(&KeyRel) Type(*Char) Len(10)
Dcl          Var(&Name)   Type(*Char) Len(40)

/*****
/* Declare the VC2EMPNAME logical file. The key for this      */
/* file is the employee first name from the VC2EMP physical  */
/* file. The key uses a shared weight sort sequence.         */
/*                                                            */
/* Declare CLF indicators.  Indicators used are &RNF and &EOF */
*****/

File         VC2EMPNAME
Inds         CLFInd(*Yes)

/*****
/* Open the file for keyed access.                            */
*****/

Open         VC2EMPNAME AccMth(*Key)

/*****
/* Attempt to read the record based on &KeyRel and &Name     */
*****/

(A) ReadRcdCLF FileID(VC2EMPNAME) Type(*Key) +
              KeyRel(&KeyRel) KeyList(&Name) +
              RcdNotFnd(&RNF) EOF(&EOF)

/*****
/* Process the result and display it                          */
*****/

Select
  When       Cond(&EOF) Then( +
              SndPgmMsg Msg('End-of-file for' +
                *BCat &KeyRel *BCat &Name))

  When       Cond(&RNF) Then( +
              SndPgmMsg Msg('Record not found for' +
                *BCat &KeyRel *BCat &Name))

  Otherwise  Cmd(SndPgmMsg Msg(&KeyRel *BCat 'of' +
                *BCat &Name *BCat 'got' *BCat +
                &EmpFName *BCat 'of' *BCat &EmpDpt))

EndSelect

/*****
/* Close the file and return to the caller                    */
*****/

Close       VC2EMPNAME
EndPgm

```

Figure 8. 5 - Source for the RNDACS CPP, RPG\_RNDACS

The program shown in Figure 8. 5 defines the two parameters &KEYREL and &NAME that the RNDACS command will pass to it, the VC2EMPNAME file, CLF indicators, and opens the file VC2EMPNAME for keyed access.

At (A) the program attempts to read one record from VC2EMPNAME where the key relation (KEYREL) and key argument (KEYLIST) are set to the values you specified with the RNDACS command. Note that while READRCDCLF is being used for the actual read operation all other commands referencing the VC2EMPNAME file are RPG-like.

If either of the conditions end-of-file or record-not-found are encountered the program sends an appropriate message. Otherwise the program sends a message indicating what record is found. Each message also includes the KEYREL and NAME values you specified with the command.

After displaying the message, the program closes the VC2EMPNAME file and returns to the caller.

The source for the program can be found in member RPG\_RNDACS of source file VC2CLF/VC2CLSRC. To create the RPG\_RNDACS program into QTEMP you use the command

```
CRTBNDCLF PGM(QTEMP/RPG_RNDACS) SRCFILE(VC2CLF/VC2CLSRC)
```

To run the program searching for a record with a key value equal to 'MATTHEW' you would use the command

```
RNDACS KEYREL(*EQ) NAME(MATTHEW)
```

You should now see the message

```
*EQ of MATTHEW got Matthew of MN
```

Using the command

```
RNDACS KEYREL(*EQ) NAME(Sam)
```

You should see the message

```
Record not found for *EQ Sam
```

as there is no employee with a first name of Sam.

Table 8.1 shows various KEYREL and NAME inputs along with the resulting message.

KEYREL	NAME	Resulting message
*EQ	MATTHEW	*EQ of MATTHEW got Matthew of MN
*LE	Matthew	*LE of Matthew got Matthew of MN
*LT	Matthew	*LT of Matthew got Kiernan of CO
*GE	Matthew	*GE of Matthew got Matthew of MN
*GT	Matthew	*GT of Matthew got Mindy of A1
*LE	A	Record not found for *LE A
*EQ	Sam	Record not found for *EQ Sam
*LE	Sam	*LE of Sam got Ruth of MN
*LT	Sam	*LT of Sam got Ruth of MN
*GE	Sam	*GE of Sam got Wayne of A1
*GT	Sam	*GT of Sam got Wayne of A1
*GT	Z	Record not found for *GT Z
*LE	R	*LE of R got Moshe of MN
*GE	R	*GE of R got Rebecca of
*GT	rebecca	*GT of rebecca got Richard of MN
*EQ	Richard	*GE of Richard got Richard of MN
*LE	Richard	*GE of Richard got Richard of MN
*LT	Richard	*LT of Richard got Rebecca of
*GE	Richard	*GE of Richard got Richard of MN
*GT	Richard	*GT of Richard got Ruth of MN
*LT	Ruth	*LT of Ruth got Richard of A1
*EQ	Alex	Record not found for *EQ Alex
*LE	Alex	Record not found for *LE Alex
*GE	Alex	*GE of Alex got Amie of A1
*LT	wayne	*LT of wayne got Ruth of MN
*GE	wayne	*GE of wayne got Wayne of A1
*GT	wayne	Record not found for *GT wayne

Table 8. 1- Inputs and results of RNDACS command

The results shown in Table 8.1 hopefully do not surprise you. Three observation however are worth pointing out.

First, when using a KEYREL of \*LE or \*GE if a record exists with the same key value then that record is returned. Otherwise the record with a key value *nearest to the*

*searched on value* is returned. These KEYREL values do not return just any record with a key value “less than” or “greater than” your search argument. They return, based on the sort sequence in effect for the index, the closest to equal match (if one exists, otherwise a record-not-found condition).

Second, in the case of ‘Richard’ where there are two records with the same first name, random access with \*EQ and ‘Richard’ returns the ‘Richard’ of department MN, random access with \*GT and ‘rebecca’ returns the ‘Richard’ of department MN, and random access with \*LT and ‘Ruth’ returns the ‘Richard’ of department A1. If you look at the relative record numbers associated with these records you will see that the MN ‘Richard’ has the lower relative record number within the based on physical file VC2EMP. The underlying relative record number has an impact on the sequence of duplicate key entries.

Third, when using random access KEYREL values you do not see any end-of-file conditions -- even when attempting to access a key value that is logically before, or after, the first and last record in the file. End-of-file conditions will not be returned by random read operations. Record-not-found is the condition you need to check for when performing random reads.

## Relative Access

RPG provides two operation codes for relative access by key – READE and READPE. This section reviews the relative access by key KEYREL values that can be used with the READRCDCLF command. The valid values are next equal to (\*NXTEQ), previous equal to (\*PRVEQ), next unique to (\*NXTUNQ), and previous unique to (\*PRVUNQ). When using the relative operators you can use the KEYLIST, KEYSTRUCT, or KEYCOUNT keywords to identify the key values. Random access KEYREL values are discussed under the topic Random access. RPG provides two operation codes for relative access by key – READE and READPE. These correspond to the READRCDCLF KEYREL values of \*NXTEQ and PRVEQ using either KEYLIST or KEYSTRUCT.

One item to keep in mind with the relative operators is that they all work relative to your current position in the file. \*NXTEQ for instance does *not* mean read the next record with a key value that is equal to the specified key. That type of request would be better satisfied with the random access operator \*EQ. Rather \*NXTEQ means: given my current position in the keyed access path, *if the next record in the keyed access path is equal to the specified key, then read the record. Otherwise, return the end-of-file condition.* To demonstrate the distinction, assume that we have an access path representing four records. The key values for the three records are AA, AA, AB, and XX. If you are currently positioned at the first record with a key value of AA, a \*NXTEQ requesting AB will not return the AB record as the next key in the access path is AA, not AB. A \*NXTEQ requesting AA however would return the second record in the access path. A subsequent \*NXTEQ for AA would then return end-of-file as all AA key values have been processed. \*PRVEQ works in the same manner, but processes the access path key values in the opposite direction.

Likewise \*NXTUNQ does *not* mean read the next record with a key value unique from the specified key. That type of request would be better satisfied with the random access operator \*GT. Rather \*NXTUNQ means: given my current position in the keyed access path, read the next record with a key value *unique from the current key value*. Following the example access path of AA, AA, AB, and XX, if you are currently positioned at the first record with a key value of AA then a \*NXTUNQ operator specifying a key value of XX will return the AB record. You will see later that the XX value does have an influence on the behavior of \*NXTUNQ, but the impact has nothing to do with the key values that are compared when using the relative operators \*NXTUNQ or \*PRVUNQ.

As was done in reviewing random access by key, one command and one program will be written to demonstrate relative access by key. Figure 8.6 shows the source for the Relative Access (RELACS) command. Figure 8.7 the source for the CPP, RPG\_RELACS.

```

CMD          PROMPT('Relative Access KEYRELS')
PARM        KWD(KEYREL) TYPE(*CHAR) LEN(10) RSTD(*YES) +
            VALUES(*NXTEQ *PRVEQ *NXTUNQ *PRVUNQ) +
            MIN(1) PROMPT('Key relation test')
PARM        KWD(NAME) TYPE(*CHAR) LEN(40) MIN(1) +
            CASE(*MIXED) PROMPT('Employee first name')

```

Figure 8. 6 - The RELACS commands source

The RELACS command is essentially the same as the previous RNDACS command of Figure 8. 4. The only changes are related to the command name and the valid values for the KEYREL keyword.

To create the Relative Access (RELACS) command into QTEMP use the command

```

CRTCMD CMD(QTEMP/RELACS) PGM(QTEMP/RPG_RELACS) +
        SRCFILE (VC2CLF/QCMDSRC)

```

```

/*****
/* Define two parameters:
/*   &KeyRel - The Key Relation (KEYREL) that is to be used
/*   &Name   - The employee name to search with in
/*             conjunction with the &KeyRel parameter
*****/

Pgm          Parm(&KeyRel &Name)

Dcl          Var(&KeyRel) Type(*Char) Len(10)
Dcl          Var(&Name)   Type(*Char) Len(40)

```

```

/*****
/* Declare the VC2EMPNAME logical file. The key for this      */
/* file is the employee first name from the VC2EMP physical  */
/* file. The key uses a shared weight sort sequence.        */
/*                                                          */
/* Declare CLF indicators.  Indicators used are &RNF and &EOF */
*****/

File      VC2EMPNAME
Inds      CLFInd(*Yes)

/*****
/* Open the file for keyed access.                          */
*****/

Open      VC2EMPNAME AccMth(*Key)

/*****
/* Attempt to read the record based on &KeyRel and &Name    */
*****/

(A1)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

/*****
/* Read by KEYLIST NAME and display the results            */
*****/

(B1)ReadRcdCLF VC2EMPNAME Type(*Key) KeyRel(&KeyRel) +
                KeyList(&Name) EOF(&EOF)

If          Cond(&EOF) Then( +
            SndPgmMsg Msg('End-of-file for' +
                *BCat &KeyRel *BCat &Name))

Else       Cmd(SndPgmMsg Msg(&KeyRel *BCat 'of' +
                *BCat &Name *BCat 'got' *BCat +
                &EmpFName *BCat 'of' *BCat &EmpDpt))

/*****
/* Attempt to read the record based on &KeyRel and &Name    */
*****/

(A2)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

/*****
/* Read by KEYLIST record field and display the results    */
*****/

(B2)ReadRcdCLF VC2EMPNAME Type(*Key) KeyRel(&KeyRel) +
                KeyList(&EmpFName) EOF(&EOF)

If          Cond(&EOF) Then( +
            SndPgmMsg Msg('End-of-file for' +
                *BCat &KeyRel *BCat &Name))

```

```

Else          Cmd(SndPgmMsg Msg(&KeyRel *BCat 'of' +
                          *BCat &Name *BCat 'got' *BCat +
                          &EmpFName *BCat 'of' *BCat &EmpDpt))

/*****
/* Attempt to read the record based on &KeyRel and &Name */
*****/

(A3)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

/*****
/* Read by KEYCOUNT and display the results */
*****/

(C) ChgVar     Var(&EmpFName) Value(XYZ)

(B3)ReadRcdCLF VC2EMPNAME Type(*Key) KeyRel(&KeyRel) +
              KeyCount(1) EOF(&EOF)

If            Cond(&EOF) Then( +
              SndPgmMsg Msg('End-of-file for' +
              *BCat &KeyRel *BCat &Name))

Else          Cmd(SndPgmMsg Msg(&KeyRel *BCat 'of' +
                          *BCat &Name *BCat 'got' *BCat +
                          &EmpFName *BCat 'of' *BCat &EmpDpt))

/*****
/* Close the file and return to the caller */
*****/

Close        VC2EMPNAME
EndPgm

```

Figure 8. 7 - Source for the RELACS CPP, RPG\_RELACS

**NOTE:** There is an intentional defect (often known as a bug) in the RPG\_RELACS example program. This defect is explained when later reviewing the results of this program.

The RPG\_RELACS CPP, on the other hand, is significantly different when compared to the RPG\_RNDACS program of Figure 8.5. The initial part of the program, defining the parameters, declaring the VC2EMPNAME file, declaring the indicators, and opening the file are the same. But things are different after that. The primary reason for the difference is that while random operators will always give you back the same result (all other things being constant anyway) for a given KEYLIST or KEYSTRUCT request, relative operators can give you back different results based on how you specify your KEYLIST, KEYSTRUCT, or KEYCOUNT request. Due to this potential for different results, RPG\_RELACS is going to perform three different types of relative reads for each use of

the RELACS command. The program will display the results for each type of relative read.

As the RELACS command is demonstrating how the relative operators work, you need to first set a position in the file that can be used as a base point for the relative operations. At (A1) the program attempts to set a position in the file by performing a random keyed read for a record where the key is equal to the &NAME value passed by the RELACS command. This is implemented using the RPG-like CHAIN command.

The program, at (B1), then attempts to read a record relative to the record read at (A1) using the READRCDCLF command. The KEYREL used is the key relation operator you selected with the RELACS KEYREL keyword. The name used for the read command, using the KEYLIST keyword, is the *name you initially provided with the RELACS NAME keyword*. RPG\_RELACS processes the results of the read and displays an appropriate message.

Having processed the record read at (B1) RPG\_RELACS now, at (A2), again performs a random read to reposition the file to a record of the file where the key value is equal to the employee first name you provided with the RELACS command. This random read is identical to the read at (A1) and will result in the same condition (either record found or not found). The program, at (B2), then attempts to read a record relative to the record read at (A2). The KEYREL used is the key relation operator you selected with the RELACS KEYREL keyword. The name used for the read command, again using the KEYLIST keyword, is in this case though the *name from the record read at (A2)*, &EMPNAME. RPG\_RELACS processes the results of the read and displays an appropriate message.

Having processed that record RPG\_RELACS then, at (A3), once again performs a random read to reposition the file to a record of the file where the key value is equal to the employee first name you provided with the RELACS command. This random read is identical to the read commands used at (A1) and (A2). The program, at (B3), then attempts to read a record relative to the record read at (A3). The KEYREL used continues to be the key relation operator you selected with the RELACS KEYREL keyword. The name used however is set to the value 'XYZ' at (C) and the *KEYLIST keyword replaced by the KEYCOUNT keyword*. The KEYCOUNT value specified is 1. A value of one indicates that the relative operation is to be based on only the 1<sup>st</sup> key field of the current record. RPG\_RELACS processes the results of the read and displays an appropriate message.

RPG\_RELACS then closes the file and returns.

To create the RPG\_RELACS CPP into QTEMP use the command

```
CRTBNDCLF PGM(QTEMP/RPG_RELACS) SRCFILE(VC2CLF/VC2CLSRC)
```

For each use of RELACS you will see three messages displayed. The first message is the result of the relative read based on the name you provided with the RELACS command, the second message the result of the relative read based on the name found in the record randomly read by key at (A2), and the third message the result of the relative read based on the name field found in the record randomly read by key after you have changed the value. Table 8.2 shows various KEYREL and NAME inputs along with the resulting messages.

KEYREL	NAME	Resulting message
*NXTEQ	Matthew	End-of-file for *NXTEQ Matthew
		End-of-file for *NXTEQ Matthew
		End-of-file for *NXTEQ Matthew
*PRVEQ	Matthew	End-of-file for *PRVEQ Matthew
		End-of-file for *PRVEQ Matthew
		End-of-file for *PRVEQ Matthew
*NXTUNQ	Matthew	*NXTUNQ of Matthew got Mindy of A1
		*NXTUNQ of Matthew got Mindy of A1
		*NXTUNQ of Matthew got Mindy of A1
*PRVUNQ	Matthew	*PRVUNQ of Matthew got Kiernan of CO
		*PRVUNQ of Matthew got Kiernan of CO
		*PRVUNQ of Matthew got Kiernan of CO
*NXTEQ	Sam	End-of-file for *NXTEQ Sam
		End-of-file for *NXTEQ Sam
		End-of-file for *NXTEQ Sam
*PRVEQ	Sam	End-of-file for *PRVEQ Sam
		End-of-file for *PRVEQ Sam
		Error message VC25021 received. Program ended as no ERR keyword or MONMSG coded
*NXTUNQ	Sam	*NXTUNQ of Sam got Amie of A1
		*NXTUNQ of Sam got Betty of RO
		*NXTUNQ of Sam got Bruce of A1
*PRVUNQ	Sam	End-of-file for *PRVUNQ Sam
		End-of-file for *PRVUNQ Sam
		End-of-file for *PRVUNQ Sam
*NXTEQ	Richard	*NXTEQ of Richard got Richard of A1
		*NXTEQ of Richard got Richard of A1
		*NXTEQ of Richard got Richard of A1
*PRVEQ	Richard	End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
*NXTUNQ	Richard	*NXTUNQ of Richard got Ruth of MN

		*NXTUNQ of Richard got Ruth of MN
		*NXTUNQ of Richard got Ruth of MN
*PRVUNQ	Richard	*PRVUNQ of Richard got Rebecca of
		*PRVUNQ of Richard got Rebecca of
		*PRVUNQ of Richard got Rebecca of
*NXTEQ	richard	End-of-file for *NXTEQ Richard
		*NXTEQ of richard got Richard of A1
		*NXTEQ of richard got Richard of A1
*PRVEQ	richard	End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ richard
*NXTUNQ	richard	*NXTUNQ of richard got Ruth of MN
		*NXTUNQ of richard got Ruth of MN
		*NXTUNQ of richard got Ruth of MN
*PRVUNQ	richard	*PRVUNQ of richard got Rebecca of
		*PRVUNQ of richard got Rebecca of
		*PRVUNQ of richard got Rebecca of

Table 8.2 - Inputs and results of the RELACS command

For the runs using a name of ‘Matthew’ the results are what you might have expected. There is only one record in the file with a first name of ‘Matthew’ so, having randomly read the first record, requests to read the next record if it has a key equal to ‘Matthew’ (\*NXTEQ) and requests to read the prior record if it has a key equal to ‘Matthew’ (\*PRVEQ) both return end-of-file.

For consistency record-not-found is also set for the READRCDCLF command but this is not the primary feedback mechanism for relative operations. When using the relative operators, if you do not specify the EOF keyword, you will be sent the escape message VC2501C - End of file for file &1. When using the relative operators you can specify the RCDNOTFND keyword with the READRCDCLF command, but you will never be sent the message VC2501B - Record not found in file &1. End-of-file is the condition to check for when using the relative operators \*NXTEQ, \*PRVEQ, \*NXTUNQ, and \*PRVUNQ.

In the case of RPG relative operations by key (READE and READPE) the RCDNOTFND keyword is not implemented while EOF is. This is due to these RPG operations only providing relative access (as contrasted with READRCDCLF which provides both relative and random access). Similarly the CHAIN command implements the RCDNOTFND keyword but not EOF.

The \*NXTUNQ and \*PRVUNQ requests, relative to ‘Matthew’s record are also what you would expect. The next unique name is ‘Mindy’, the previous unique name is ‘Kiernan’.

Test runs with 'Sam', on the other hand, seem to be returning rather strange results. In the runs using \*NXTEQ you see end-of-file being returned and, in two out of three cases, \*PRVEQ also returns end-of-file. In the third case though, using \*PRVEQ with KEYCOUNT, the READRCDCLF command receives the error message VC25021 - The &4 command was not successful when accessing File ID &2. Moving on to \*NXTUNQ requests, you see the records being read apparently progressing through the access path with the first READRCDCLF \*NXTUNQ reading 'Amie', the next READRCDCLF reading 'Betty', and the final READRCDCLF reading 'Bruce'. None of which are correct as logically 'Wayne' would be the next unique key value after 'Sam'. \*PRVUNQ appears to be consistent with end-of-file being returned, but you might have expected 'Ruth' to be read as she represents the unique key previous to a value of 'Sam'.

Looking at the results for 'Sam' you can perhaps rationalize what the system is doing, but don't bother. What you are seeing is, while not random, results that are not dependable. The KEYREL relative operators are all dependent on your current position in the file. As there is no record in the file with a key value of 'Sam' all of the random reads (those done at (A1), (A2), and (A3) of RPG\_RELACS) returned a record-not-found condition. When you do not have a known position in the file (record-not-found being an instance of not knowing for sure your position) you should not use relative requests until you have established a known position in the file. Requesting the next (or previous) record, relative to a record that does not exist, does not make a lot of sense when you think about it. This is a *defect* and can cause your application program to work in a highly undependable manner. In the provided RPG\_RELACS sample program, the READRCDCLF commands at (B1), (B2), and (B3) should only be run when condition &RNF (from the READRCDCLF command at (A1), (A2), or (A3)) is off (not true) – indicating that the random read was successful. One way to correct this defect is shown later with the program RPG\_RELPOS.

After 'Sam', the RELACS command is run with the names 'Richard' and 'richard'. 'Richard', as you might recall, is the one first name in the VC2EMP database file with two records. One 'Richard' works in department MN, the other in department A1.

As we saw with the RNDACS command and Table 8.1, when randomly reading a record using the name of 'Richard' (as RPG\_RELACS does at (A1), (A2), and (A3)) the record read is always the record for 'Richard' of department MN. This can also be inferred with Table 8.2 as all three \*NXTEQ read requests return 'Richard' of department A1 and all three \*PRVEQ read requests return end-of-file. When working with 'Richard' the \*NXTUNQ and \*PRVUNQ operations also appear to be working as expected. In all cases 'Ruth' is returned as the next unique key value and 'Rebecca' as the previous unique key value.

Reviewing the results for a search argument of 'richard' (rather than 'Richard') with the RELACS command you do however see some perhaps unexpected results.

The first \*NXTEQ returns end-of-file while the other two \*NXTEQ requests return the record for 'Richard' of department A1. When using either \*NXTEQ (READE) or \*PRVEQ (READPE) and either KEYLIST or KEYSTRUCT with READRCDCLF the test for equality is done based on the *exact* name specified with the KEYLIST or KEYSTRUCT keyword and the *exact* name as it is stored in the database record. So while the initial random read using \*EQ at (A1) succeeded in finding 'Richard' with a search argument of 'richard' (due to the SRTSEQ specification when creating the VC2EMPNAME logical file) the \*NXTEQ at (B1), using the value 'richard', failed to compare as being equal due to the &EMPFNAME field of the second record being 'Richard'. At (B2) however the program is using the value of &EMPFNAME as read from the initial record at (A2). This value is 'Richard' and so compares as equal to the value of 'Richard' found in the department A1 record.

For the third \*NXTEQ, using KEYCOUNT at (B3), no value is actually supplied by you. This comparison for equality of key fields is based strictly on the key values (not the data values as is the case when KEYLIST or KEYSTRUCT are used) of the current record. Even though the program has changed at (C) the value of the &EMPFNAME field for the record initially read at (A3) the record for 'Richard' in department A1 still compares as equal. This would not have been the case had you changed the value of &EMPFNAME prior to (B2). Because the comparison when you specify KEYCOUNT is done by key value, as opposed to record data with KEYLIST or KEYSTRUCT, this also means that the SRTSEQ specification for the key fields is in effect. That is, if the name 'Richard' of department A1 had actually been stored in the database as 'RICHARD', using KEYCOUNT would have allowed search arguments of 'Richard', 'richard', 'RiChArD', etc to all compare as equal. This is not the case when using KEYLIST or KEYSTRUCT.

## Positioning examples

### Random positioning

As with the case of RPG providing random reads with the CHAIN operation while the READRCDCLF command provides many more options, so too does CLF provide many more options for random positioning than just the SETLL and SETGT RPG-like commands.

When positioning to records by key you can position either randomly or relative to your current location in the file. This section reviews the POSDBFCLF KEYREL random positioning by key values. The valid values are equal to (\*EQ), less than or equal to (\*LE), greater than or equal to (\*GE), less than (\*LT), and greater than (\*GT). With the relational operators you can only use the KEYLIST or the KEYSTRUCT keywords to identify the key values. The KEYCOUNT keyword is not supported. The RPG SETLL operation is effectively a POSDBFCLF KEYREL value of \*GE and the SETGT operation a KEYREL of \*GT.

In keeping with the previous discussion of READRCDCLF one command and one program will be written. The command will prompt for a valid KEYREL value and the first name of an employee. The program will be the CPP for the command and define two parameters – &KEYREL and &NAME. These parameters will then be used to specify the KEYREL value you want to test and the employee first name you want to randomly position to. Figure 8.8 shows the source for the command Random Positioning (RNDPOS) and Figure 8.9 the source for the program (RPG\_RNDPOS).

```
CMD          PROMPT('Random Positioning KEYRELS')
PARM         KWD(KEYREL) TYPE(*CHAR) LEN(10) RSTD(*YES) +
              VALUES(*EQ *LE *GE *LT *GT) MIN(1) +
              PROMPT('Key relation test')
PARM         KWD(NAME) TYPE(*CHAR) LEN(40) MIN(1) +
              CASE(*MIXED) PROMPT('Employee first name')
```

Figure 8. 8 - The RNDPOS command source

To create the RNDPOS command into QTEMP use the command

```
CRTCMD CMD(QTEMP/RNDPOS) PGM(QTEMP/RPG_RNDPOS) +
        SRCFILE(VC2CLF/QCMDSRC)
```

```
/* **** */
/* Define two parameters: */
/* &KeyRel - The Key Relation (KEYREL) that is to be used */
/* &Name - The employee name to search with in */
/* conjunction with the &KeyRel parameter */
/* **** */
```

```

Pgm          Parm(&KeyRel &Name)

Dcl          Var(&KeyRel) Type(*Char) Len(10)
Dcl          Var(&Name)   Type(*Char) Len(40)

/*****
/* Declare the VC2EMPNAME logical file. The key for this      */
/* file is the employee first name from the VC2EMP physical  */
/* file. The key uses a shared weight sort sequence.         */
/*                                                           */
/* Declare CLF indicators. Indicators used are &RNF and &EOF */
*****/

File         VC2EMPNAME
Inds         CLFInd(*Yes)

/*****
/* Open the file for keyed access.                          */
*****/

Open         VC2EMPNAME AccMth(*Key)

/*****
/* Attempt to position to the requested record              */
*****/

(A) PosDBFCLF FileID(VC2EMPNAME) Type(*Key) +
           KeyRel(&KeyRel) KeyList(&Name) +
           RcdNotFnd(&RNF)

If           Cond(&RNF) Then(SndPgmMsg Msg( +
           'Record not found for' *BCat +
           &KeyRel *BCat &Name))

Else        Cmd(Do)
(B)       Read VC2EMPNAME EOF(&EOF)

           /*****
           /* Process the result and display it                */
           *****/

           If Cond(&EOF) Then( +
           SndPgmMsg Msg('End-of-file for' +
           *BCat &KeyRel *BCat &Name))

           Else Cmd(SndPgmMsg Msg(&KeyRel *BCat +
           'of' *BCat &Name *BCat +
           'got' *BCat &EMPName *BCat +
           'of' *BCat &EmpDpt))

           EndDo

/*****
/* Close the file and return to the caller                    */
*****/

```

```

/*****/

Close      VC2EMPNAME
EndPgm

```

Figure 8. 9 - Source for the RNDPOS CPP, RPG\_RNDPOS

The RPG\_RNDPOS program is very similar to what you saw in Figure 8. 5 - Source for the RNDACS CPP, RPG\_RNDACS. The major change is that the READRCDCLF command is replaced at (A) with the POSDBFCLF command followed by a conditional RPG-like READ at (B).

The POSDBFCLF command is written with the RCDNOTFND keyword. If a record-not-found condition is returned by the positioning request then the READ command is not run. If you were to run it you would find that it always returns end-of-file.

The READ command is written with the EOF keyword. As READ is a command that works relative to the current position, the READ command uses EOF rather than RCDNOTFND to detect when there are no records to be read. The READ command is run so that you can visually determine your current position within the file.

To create the RPG\_RNDPOS program into QTEMP you can use the command

```
CRTBNDCLE PGM(QTEMP/RPG_RNDPOS) SRCFILE(VC2CLF/VC2CLSRC)
```

Running the command RNDPOS, with the same keyword values as were used for Table 8. 1- Inputs and results of RNDACS command, you get the results shown in Table8.3.

KEYREL	NAME	Resulting message
*EQ	MATTHEW	*EQ of MATTHEW got Matthew of MN
*LE	Matthew	*LE of Matthew got Matthew of MN
*LT	Matthew	*LT of Matthew got Kiernan of CO
*GE	Matthew	*GE of Matthew got Matthew of MN
*GT	Matthew	*GT of Matthew got Mindy of A1
*LE	A	Record not found for *LE A
*EQ	Sam	Record not found for *EQ Sam
*LE	Sam	*LE of Sam got Ruth of MN
*LT	Sam	*LT of Sam got Ruth of MN
*GE	Sam	*GE of Sam got Wayne of A1
*GT	Sam	*GT of Sam got Wayne of A1
*GT	Z	Record not found for *GT Z

*LE	R	*LE of R got Moshe of MN
*GE	R	*GE of R got Rebecca of
*GT	rebecca	*GT of rebecca got Richard of MN
*EQ	Richard	*GE of Richard got Richard of MN
*LE	Richard	*GE of Richard got Richard of MN
*LT	Richard	*LT of Richard got Rebecca of
*GE	Richard	*GE of Richard got Richard of MN
*GT	Richard	*GT of Richard got Ruth of MN
*LT	Ruth	*LT of Ruth got Richard of A1
*EQ	Alex	Record not found for *EQ Alex
*LE	Alex	Record not found for *LE Alex
*GE	Alex	*GE of Alex got Amie of A1
*LT	wayne	*LT of wayne got Ruth of MN
*GE	wayne	*GE of wayne got Wayne of A1
*GT	wayne	Record not found for *GT wayne

Table 8. 3 - Inputs and results of the RNDPOS command

The results of RNDACS and RNDPOS are the same. Running POSDBFCLF with a random positioning KEYREL value followed by a READ command is equivalent to running READRCDFCLF with a random access KEYREL value.

### Relative positioning

There are no equivalent free form RPG operation codes to the functions reviewed in this section.

This section reviews the relative positioning by key KEYREL values that can be used with the POSDBFCLF command. The valid values are next equal to (\*NXTEQ), previous equal to (\*PRVEQ), next unique to (\*NXTUNQ), and previous unique to (\*PRVUNQ). When using the relative operators you can use the KEYLIST, KEYSTRUCT, or KEYCOUNT keywords to identify the number of key values to use when comparing the current record to the requested position. A key distinction between relative access with READRCDFCLF and relative positioning with POSDBFCLF is that positioning is *always done based on the key values of the record currently positioned to*. The values that you specify when using the KEYLIST or KEYSTRUCT keywords are not considered. These keywords are, in the case of relative positioning, only identifying the number of fields to test. Random positioning KEYREL values are discussed under the topic Random positioning.

Figure 8.10 shows the source for the Relative Positioning KEYRELS (RELPOS) command, Figure 8.11 the source for RELPOS's CPP RPG\_RELPOS, and Table 8.4 the results of running RELPOS with the same parameter values as were previously used running RELACS.

```

CMD          PROMPT('Relative Positioning KEYRELS')
PARM        KWD(KEYREL) TYPE(*CHAR) LEN(10) RSTD(*YES) +
            VALUES(*NXTEQ *PRVEQ *NXTUNQ *PRVUNQ) +
            MIN(1) PROMPT('Key relation test')
PARM        KWD(NAME) TYPE(*CHAR) LEN(40) MIN(1) +
            CASE(*MIXED) PROMPT('Employee first name')

```

Figure 8. 10 - The RELPOS command source

To create the RELPOS command into QTEMP use the command

```

CRTCMD CMD(QTEMP/RELPOS) PGM(QTEMP/RPG_RELPOS) +
        SRCFILE(VC2CLF/QCMDSRC)

```

```

/*****/
/* Define two parameters: */
/* &KeyRel - The Key Relation (KEYREL) that is to be used */
/* &Name - The employee name to search with in */
/* conjunction with the &KeyRel parameter */
/*****/

Pgm          Parm(&KeyRel &Name)

Dcl          Var(&KeyRel) Type(*Char) Len(10)
Dcl          Var(&Name) Type(*Char) Len(40)

/*****/
/* Declare the VC2EMPNAME logical file. The key for this */
/* file is the employee first name from the VC2EMP physical */
/* file. The key uses a shared weight sort sequence. */
/* */
/* Declare CLF indicators. Indicators used are &RNF and &EOF */
/*****/

File         VC2EMPNAME
Inds         CLFInd(*Yes)

/*****/
/* Open the file for keyed access. */
/*****/

Open         VC2EMPNAME AccMth(*Key)

/*****/
/* Attempt to read the record based on &KeyRel and &Name */
/*****/

```

```

(A1)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

(B1)If         Cond(*Not &RNF) Then(Do)

                /*****
                /* Position by KEYLIST NAME and display results */
                *****/

                PosDBFCLF FileID(VC2EMPNAME) Type(*Key) +
                        KeyRel(&KeyRel) KeyList(&Name) +
                        EOF(&EOF)

                If Cond(*Not &EOF) Then(Do)
                    Read VC2EMPNAME EOF(&EOF)

                    If Cond(&EOF) Then(SndPgmMsg Msg( +
                        'End-of-file for' +
                        *BCat &KeyRel *BCat &Name))

                    Else Cmd(SndPgmMsg Msg( +
                        &KeyRel *BCat 'of' *BCat &Name +
                        *BCat 'got' *BCat &EmpFName +
                        *BCat 'of' *BCat &EmpDpt))

                    EndDo

                Else Cmd(SndPgmMsg Msg('End of file for' +
                    *BCat &KeyRel *BCat &Name))

                EndDo

                Else Cmd(SndPgmMsg Msg('Record not found for' +
                    *BCat &KeyRel *BCat &Name))

                /*****
                /* Attempt to read the record based on &KeyRel and &Name */
                *****/

(A2)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

(B2)If         Cond(*Not &RNF) Then(Do)

                /*****
                /* Position by KEYLIST &EmpFName and show results */
                *****/

                PosDBFCLF FileID(VC2EMPNAME) Type(*Key) +
                        KeyRel(&KeyRel) KeyList(&EmpFName) +
                        EOF(&EOF)

                If Cond(*Not &EOF) Then(Do)
                    Read VC2EMPNAME EOF(&EOF)

```

```

        If Cond(&EOF) Then(SndPgmMsg Msg( +
            'End-of-file for' +
            *BCat &KeyRel *BCat &Name))

        Else Cmd(SndPgmMsg Msg( +
            &KeyRel *BCat 'of' *BCat &Name +
            *BCat 'got' *BCat &EmpFName +
            *BCat 'of' *BCat &EmpDpt))

        EndDo

    Else Cmd(SndPgmMsg Msg('End of file for' +
        *BCat &KeyRel *BCat &Name))

    EndDo

Else      Cmd(SndPgmMsg Msg('Record not found for' +
    *BCat &KeyRel *BCat &Name))

/*****
/* Attempt to read the record based on &KeyRel and &Name      */
/*****

(A3)Chain      &Name VC2EMPNAME RcdNotFnd(&RNF)

(B3)If          Cond(*Not &RNF) Then(Do)

/*****
/* Position by KEYCOUNT and display results      */
/*****

ChgVar Var(&EmpFName) Value(XYZ)

PosDBFCLF FileID(VC2EMPNAME) Type(*Key) +
    KeyRel(&KeyRel) KeyCount(1) +
    EOF(&EOF)

If Cond(*Not &EOF) Then(Do)
    Read VC2EMPNAME EOF(&EOF)

    If Cond(&EOF) Then(SndPgmMsg Msg( +
        'End-of-file for' +
        *BCat &KeyRel *BCat &Name))

    Else Cmd(SndPgmMsg Msg( +
        &KeyRel *BCat 'of' *BCat &Name +
        *BCat 'got' *BCat &EmpFName +
        *BCat 'of' *BCat &EmpDpt))

    EndDo

Else Cmd(SndPgmMsg Msg('End of file for' +
    *BCat &KeyRel *BCat &Name))

```

```

                                EndDo

Else      Cmd(SndPgmMsg Msg('Record not found for' +
                                *BCat &KeyRel *BCat &Name))

/*****
/* Close the file and return to the caller      */
/*****

Close      VC2EMPNAME
EndPgm

```

Figure 8. 11- Source for the RELPOS CPP, RPG\_RELPOS

The RPG\_RELPOS program is essentially the RPG\_RELACS program you saw earlier *with the defect that was discussed at that time removed*. The other changes made are the same as you saw when moving from RPG\_RNDACS to RPG\_RNDPOS – replacing the random read READRCDCLF command with a random positioning POSDBFCLF command followed by a READ command. With RPG\_RELPOS you use a relative access POSDBFCLF command again followed by a READ command.

Removing the defect from RPG\_RELACS when writing RPG\_RELPOS is simple. It’s a matter of testing the condition of variable &RNF after the CHAINs of (A1), (A2), and (A3) have completed. If &RNF is true, indicating that no record was found, the program displays a ‘record not found’ message and continues to the next test case. If &RNF is false ((B1), (B2), and (B3)), indicating that a record was found that meets the search criteria, then the program runs the POSDBFCLF command with a relative operator. With this change it is now safe to request a next, or previous, positioning request as we have a current record in the file upon which to base such a request.

To create the RPG\_RELPOS CPP into QTEMP you use the command

```
CRTBNDCLE PGM(QTEMP/RPG_RELPOS) SRCFILE(VC2CLF/VC2CLSRC)
```

Table 8.4 shows the resulting messages when running the RELPOS command with the same test values as you used with the RELACS command.

KEYREL	NAME	Resulting message
*NXTEQ	Matthew	End-of-file for *NXTEQ Matthew
		End-of-file for *NXTEQ Matthew
		End-of-file for *NXTEQ Matthew
*PRVEQ	Matthew	End-of-file for *PRVEQ Matthew
		End-of-file for *PRVEQ Matthew
		End-of-file for *PRVEQ Matthew
*NXTUNQ	Matthew	*NXTUNQ of Matthew got Mindy of A1

		*NXTUNQ of Matthew got Mindy of A1
		*NXTUNQ of Matthew got Mindy of A1
*PRVUNQ	Matthew	*PRVUNQ of Matthew got Kiernan of CO
		*PRVUNQ of Matthew got Kiernan of CO
		*PRVUNQ of Matthew got Kiernan of CO
*NXTEQ	Sam	Record not found for *NXTEQ Sam
		Record not found for *NXTEQ Sam
		Record not found for *NXTEQ Sam
*PRVEQ	Sam	Record not found for *PRVEQ Sam
		Record not found for *PRVEQ Sam
		Record not found for *PRVEQ Sam
*NXTUNQ	Sam	Record not found for *NXTUNQ Sam
		Record not found for *NXTUNQ Sam
		Record not found for *NXTUNQ Sam
*PRVUNQ	Sam	Record not found for *PRVUNQ Sam
		Record not found for *PRVUNQ Sam
		Record not found for *PRVUNQ Sam
*NXTEQ	Richard	*NXTEQ of Richard got Richard of A1
		*NXTEQ of Richard got Richard of A1
		*NXTEQ of Richard got Richard of A1
*PRVEQ	Richard	End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
*NXTUNQ	Richard	*NXTUNQ of Richard got Ruth of MN
		*NXTUNQ of Richard got Ruth of MN
		*NXTUNQ of Richard got Ruth of MN
*PRVUNQ	Richard	*PRVUNQ of Richard got Rebecca of
		*PRVUNQ of Richard got Rebecca of
		*PRVUNQ of Richard got Rebecca of
*NXTEQ	richard	*NXTEQ of richard got Richard of A1
		*NXTEQ of richard got Richard of A1
		*NXTEQ of richard got Richard of A1
*PRVEQ	richard	End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ Richard
		End-of-file for *PRVEQ richard
*NXTUNQ	richard	*NXTUNQ of richard got Ruth of MN
		*NXTUNQ of richard got Ruth of MN
		*NXTUNQ of richard got Ruth of MN
*PRVUNQ	richard	*PRVUNQ of richard got Rebecca of
		*PRVUNQ of richard got Rebecca of

		*PRVUNQ of richard got Rebecca of
--	--	-----------------------------------

Table 8. 4 - Inputs and results of the RELPOS command

For all of the tests based on ‘Matthew’ you see the same results you did with the RELACS command.

For the tests involving ‘Sam’ there are differences however. With the test for a successful read after attempting to randomly read the first record for ‘Sam’, (B1), (B2), and (B3), you now see ‘record not found’ messages rather than incorrect names in some cases and error messages in other cases.

As with the test cases for ‘Matthew’, the test cases using a RELPOS NAME value of ‘Richard’ also results in the same messages being displayed.

The test case with ‘richard’ however is different. With RELACS the first message showed ‘End-of-file for \*NXTEQ Richard’, with RELPOS the first message is now ‘\*NXTEQ of richard got Richard of A1’. RELPOS successfully matched the ‘Richard’ of department A1 with the ‘richard’ of the search argument. This difference in behavior is because POSDBFCLF, when used with the relative KEYREL requests, always uses the key *as it is stored in the access path* and the key *as it exists in the current record*. The values you specify for KEYLIST or KEYSTRUCT are used only to determine how many fields of the key should be compared. Not what values should be used for the comparison. So POSDBFCLF, using KEYREL(\*NXTEQ) and KEYLIST(&NAME), is able to take advantage of the SRTSEQ table used when creating the logical file VC2EMPNAME. READRCDCLF, using KEYREL(\*NXTEQ) and KEYLIST(&NAME), is not able to take advantage of the SRTSEQ table. As mentioned previously, READRCDCLF in this situation must do a comparison based on the &NAME you specify for the KEYLIST keyword and the data as it exists in the next record of the file.

## EXAMPLE USING DATABASE READ, DELETE, AND UPDATE WITH THE CLF PRECOMPILER

In addition to reading, or positioning to, records from a file you can also write new records using the Write New Records (WRITE), update records using the Modify Existing Record (UPDATE) command, and delete records using the Delete Record (DELETE) command.

Figure 8. 12 provides the source for a very simple program that demonstrates:

- reading a record by key
- deleting a record
- updating a record
- detecting a record-not-found condition when attempting to randomly read a record

```
Pgm          Parm(&EmpNo &Action)

/*****
/*
/* This program takes two parameters:
/*   &EmpNo - Employee number
/*   &Action - Action to be taken on employee record identified
/*             by the first parameter.  The valid actions are:
/*             D - Delete the employee record
/*             I - Update the employee record to Inactive
/*             P - Update the employee record to Part time
/*
/* Any other &Action value will result in an error message
/*
/* The program does not use an interactive interface as this
/* example is focused on database access.  The Using Display
/* Files chapter provides a more functional version of this
/* program using a display file.
/*
/*
*****/

/*****
/* Define the parameters.
/*
/* &EmpNo is declared as *DEC with (15 5) as it is being passed
/* as a numeric input parameter from the command line
*****/
```

```

Dcl      Var(&EmpNo)      Type(*Dec)  Len(15 5)
Dcl      Var(&Action)     Type(*Char) Len(1)

/*****
/* Declare the VC2EMP file.                               */
*****/

File      VC2EMP

/*****
/* Declare CLF-related indicators                         */
*****/

Inds      CLFInd(*Yes)

/*****
/* Test to determine if both paramters were passed      */
*****/

CHGVAR    VAR(&Action) VALUE(&Action)
MONMSG    MSGID(MCH3601) EXEC(DO)
          RMVMSG CLEAR(*ALL)
          SNDPGMMSG MSG( +
          'There are two required parameters: EmpNo and Action')
          RETURN
          ENDDO

/*****
/* Open the VC2EMP file for keyed access and read/write/update */
*****/

Open      VC2EMP Usage(*Both) AccMth(*KEY)

/*****
/* Read the employee record with a key equal to &EmpNo.  If no */
/* record is found, turn on logical variable &RNF           */
*****/

Chain     &EmpNo VC2EMP RcdNotFnd(&RNF)

/*****
/* If no employee record is found then send a message to the caller */
*****/

If        Cond(&RNF) Then( +
          SndPgmMsg Msg('Employee not found.'))

/*****
/* Otherwise determine what action has been requested and do it */
*****/

Else      Cmd(Select)

          When Cond(&Action = D) Then( +

```

```

        Delete VC2EMP)

    When Cond(&Action = I) Then(Do)
        ChgVar Var(&EmpSts) Value(I)
        Update VC2EMP
        EndDo

    When Cond(&Action = P) Then(Do)
        ChgVar Var(&EmpSts) Value(P)
        Update VC2EMP
        EndDo

    Otherwise Cmd( +
        SndPgmMsg Msg('Action' *BCat &Action *BCat +
            'is not supported.')
```

```

    EndSelect

/*****
/* Close the file and return
/*****

    Close      VC2EMP

    EndPgm

```

Figure 8. 12 - The RPG\_MNTEMP program

The source for this program can be found in member RPG\_MNTEMP (Maintain Employees) of source file VC2CLSRC in library VC2CLF.

The RPG\_MNTEMP program is written to accept two input parameters – an employee number and an action code. The employee number identifies the employee to be maintained, the action code the action to perform on the employee. An action code of I indicates that the employee status should be set to Inactive, an action code of P that the employee status should be set to Part time, and an action code of D that the employee record should be deleted from the file.

To create the RPG\_MNTEMP program into QTEMP you use the command

```
CRTBNDCLF PGM(QTEMP/RPG_MNTEMP) SRCFILE(VC2CLF/VC2CLSRC)
```

Before running the program verify that the contents of the VC2EMP physical file are at a known state. First run the DSPPFM VC2CLF command. You should see an entry for 'Wayne' with the letter 'I', for Inactive, just before the 'W' of 'Wayne'. You should also see an entry for 'Ruth' with an 'I', for Inactive, just before the 'R' of 'Ruth'. If you do not see these two records as described refer to Appendix B. VC2EMP and VC2DPT Sample Files to reload the VC2EMP file.

From a command line run the command

```
CALL PGM(QTEMP/RPG_MNTEMP) PARM(53 P)
```

Where there is a space between the '53' and the 'P'. This should cause the entry for 'Wayne' (employee number 53) to now show 'P' (for Part time) where you previously saw an I. You can confirm that the database update succeeded by using the command DSPPFM VC2EMP.

Now from a command line enter:

```
CALL PGM(QTEMP/RPG_MNTEMP) PARM(52 D)
```

This should delete the entry for 'Ruth'. Again you can confirm that 'Ruth' is now gone with DSPPFM VC2EMP.

And as a third test case you can use:

```
CALL PGM(RPG_MNTEMP) PARM(2 I)
```

You should see the message 'Employee not found' as there is no employee with an employee number of 2.

## USING OPNQRYPF

CLF supports the Open Query File (OPNQRYPF) command enabling access to sets of database records that satisfy a database query request. The considerations for using OPNQRYPF are the same as you find with other languages. You need to share the open data path with the Override Database File (OVRDBF) command, run the OPNQRYPF command, open the file using the CLF OPEN command, and then process the file as you would any other database file.

### *OPNQRYPF with Record Selection*

Figure 8. 13 shows a CLF application program that uses the OPNQRYPF command to dynamically select employees from the VC2EMP employee database that meet a given employment status.

```
/******  
/* This program displays the employees with a status of */  
/* the parameter passed to the program. If no parameter */  
/* is provided the program defaults to a status of F. The */  
/* valid status values are F, P, and I. Other values can */  
/* be used but will result in no employees being displayed*/  
/* */  
/* The program uses the OPNQRYPF command to select the */  
/* employees. */  
/******  
  
Pgm          Parm(&Status_In)  
Dcl          Var(&Status_In) Type(*Char) Len(1)  
Dcl          Var(&Status)   Type(*Char) Len(1)  
  
/******  
/* Declare the file definition that will be used (VC2EMP) */  
/* and the &EOF indicator. */  
/******  
  
File         VC2EMP  
Dcl          Var(&EOF)      Type(*Lgl)  
  
/******  
/* Set the employee status to be selected based on the */  
/* input parameter value. If no input value, use 'F' */  
/******  
  
ChgVar       Var(&Status) Value(&Status_In)  
MonMsg       MsgID(MCH3601) Exec( +  
             ChgVar Var(&Status) Value(F))
```

```

/*****
/* Override VC2EMP to share the open data path, run the      */
/* OPNQRYF command, and open the VC2EMP CLF file            */
/*****

OvrDBF      VC2EMP Share(*Yes)
OpnQryF     VC2EMP QrySlt( +
            'EmpSts *Eq "' *Cat &Status *Cat "' ) +
            KeyFld((EmpFname))
Open        VC2EMP AccMth(*Key)

/*****
/* Display a title for the list to follow                    */
/*****

SndPgmMsg  Msg('Listing of Employees with Status' +
              *BCat &Status)

/*****
/* Read all records in the file, displaying the employees   */
/* with their names indented two spaces                      */
/*****

Read       VC2EMP EOF(&EOF)
DoWhile    Cond(*Not &EOF)
           SndPgmMsg  Msg(' ' *Cat &EmpFname)
           Read VC2EMP EOF(&EOF)
           EndDo

/*****
/* Display and end of report line, close the files, and    */
/* end the program.                                        */
/*****

SndPgmMsg  Msg('End of Employee Listing')

Close      VC2EMP
CloF       VC2EMP
EndPgm

```

Figure 8. 13 - Using OPNQRYF for record selection

The source for program RPG\_EMPSTS can be found in member RPG\_EMPSTS of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use

```
CRTBNDCLF PGM(QTEMP/RPG_EMPSTS) SRCFILE(VC2CLF/VC2CLSRC)
```

To display all parttime employees you can

```
CALL PGM(QTEMP/RPG_EMPSTS) PARM(P)
```

Note that the &STATUS\_IN parameter of the RPG\_EMPSTS program is only optional if you compile the application as ILE. For OPM programs the system will require that you pass the parameter.

### **OPNQRYF with Join, Record Grouping, and Mapped Fields**

Figure 8. 14 shows a CLF application program that uses the OPNQRYF command to join the VC2DPT and VC2EMP sample databases by department ID, group employees by department, and then count the number of employees in each department. The count of employees per department is then shown in department ID sequence.

```

/*****
/* This program displays the number of employees in each */
/* department. If there are no employees in a department */
/* the department does not appear in the listing.        */
/*                                                      */
/* The program uses the OPNQRYF command to count the   */
/* number of employees in each department.              */
*****/

      Pgm

/*****
/* While EMPCOUNT in file VC2EMPCNT is defined as numeric */
/* the program declares it as character. The precompiler  */
/* accepts this redefinition. This is done to demonstrate */
/* flexibility in data types and also to avoid having to  */
/* use CHGVAR from EMPCOUNT to a character field so that */
/* the count of employees can be used in the SNDPGMMSG    */
/* command.                                               */
*****/

      (A) Dcl          Var (&EmpCount)   Type (*Char) Len (2)

/*****
/* Declare the file definition that will be used          */
/* (VC2EMPCNT) and CLF related indicators                */
*****/

      (B) File        Anything  FILE (VC2EMPCNT)
          Inds         CLFInd (*Yes)

/*****
/* Override VC2DPT to share the open data path, run the  */
/* OPNQRYF command, and open the VC2DPT CLF file         */
*****/

      OvrDBF         VC2DPT Share (*Yes)
      (C) OpnQryF    File ((VC2DPT) (VC2EMP)) Format (VC2EMPCNT) +
                    KeyFld ((DptID)) JFld ((1/DptID 2/EmpDpt)) +

```

```

                GrpFld(DptID DptName) +
                MapFld((EmpCount '%count'))
(D) Open      Anything File(VC2DPT) AccMth(*Key)

/*****
/* Display a title for the list to follow          */
*****/

        SndPgmMsg  Msg('Count of Employees by Department')

/*****
/* Read all records in the file, displaying the    */
/* department ID, department name, and number of  */
/* employees*/
*****/

        Read      Anything EOF(&EOF)
        DoWhile   Cond(*Not &EOF)
                SndPgmMsg Msg('  ' *Cat +
                &DptID *BCat &DptName *Cat &EmpCount)
                Read Anything EOF(&EOF)
        EndDo

/*****
/* Display an end of report line, close the files, and */
/* end the program.                                     */
*****/

        SndPgmMsg  Msg('End of Departments with Employees')

        Close      Anything
        CloF       VC2DPT
        DltOvr     VC2DPT
        EndPgm

```

Figure 8. 14 - OPNQRYP with Join, Record Grouping, and Mapping

At (B) of Figure 8. 14 the RPG\_EMPCNT program uses the FILE command to declare the file VC2EMPCNT using a FILEID name of Anything. The VC2EMPCNT file is used for the FILE keyword of the FILE command as that is the format defined by the OPNQRYP command at (C) with the FORMAT keyword. The file VC2EMPCNT represents the format of the data that will be read by the program from the query file generated by the OPNQRYP command. A FILEID of Anything is used with the FILE command to demonstrate that the FILEID is not used other than as a reference point for other CLF commands. The program at (D) opens the file with a FILE keyword value of VC2DPT. This value is selected as it is the first file name used by the OPNQRYP command with the FILE keyword at (C).

The VC2EMPCNT file definition is shown in Figure 8. 15.

```

.....A.....T.Name+++++RLen++TDpB.....Functions+++++
R  EMPCNTRCD
   DPTID           2
   DPTNAME        30
   EMPCOUNT       2  0

```

Figure 8. 15 - DDS for the VC2EMPCNT query file

The field EMPCOUNT is defined as a 2-digit packed decimal value and is the OPNQRYP mapping of the count (%count) of employees in each department. As this count will be displayed by RPG\_EMPCNT using the SNDPGMMSG command the program at (A) defines the CL variable &EMPCOUNT as a 2-byte character field prior to declaring the file at (B). This is done so that the employee count value can be used directly as part of the MSG text sent using the SNDPGMMSG command. The alternative, if not using the precompiler, would be to define a 2-byte character field and use CHGVAR to change the value of this field to the value of &EMPCOUNT. This new field would then be used as part of the MSG text. Using the precompiler you can avoid this type of unnecessary data type conversion.

Having declared the VC2EMPCNT file (and CLF related indicators) RPG\_EMPCNT

- overrides the query file VC2DPT to share its open data path
- runs the OPNQRYP command at (C) to populate the query file VC2DPT with the results of a join using VC2DPT and VC2EMP
- opens the VC2DPT query file
- writes a message identifying the listing about to be shown
- reads all records found in the query file displaying each department ID, name, and employee count
- sends a message indicating the list is finished
- closes the files, deletes the override, and exits.

The source for RPG\_EMPCNT can be found in member RPG\_EMPCNT of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use the command

```
CRTBNDCLF PGM(QTEMP/RPG_EMPCNT) SRCFILE(VC2CLF/VC2CLSRC)
```

## COMMITMENT CONTROL

When processing a file you can also use commitment control to treat multiple file operations as an all or nothing set of database transactions. With commitment control, you are ensured one of two outcomes for the file operations:

- All of the file operations are successful (a commit operation)
- None of the file operations have any effect (a rollback operation)

In this way you process a group of file operations as a discrete unit. To open a file under commitment control you specify CMTCTL(\*YES) on the OPEN command.

The commitment control provided with CLF is integrated with the commitment control of the i operating system. To establish a commitment control definition you use the Start Commitment Control (STRCMTCTL) CL command defining the desired lock level (LCKLVL), commitment scope (CMTSCOPE), etc. To commit or rollback database transactions you use the traditional COMMIT and ROLLBACK CL commands. It's just that now the committed, or rolled back, transaction includes CLF based database changes in addition to your RPG, COBOL, C, and SQL based changes. An example of using commitment control can be found in source member RPG\_CMTCTL of source file VC2CLF/VC2CLSRC.

## TRIGGER PROGRAMS

If you develop both application and trigger programs using CLF then you may want to create the CLF trigger programs as ILE CL programs using an explicitly named activation group. Having the trigger program run in its own activation group can prevent recursion when an application program uses a command such as Modify Existing Record (UPDATE) and an update trigger program is subsequently called where the trigger program also uses the UPDATE command.

## DIFFERENCES FROM CONVENTIONAL CL FILE USAGE

Beyond the obvious differences of being able to update, write, randomly read, and delete records using CLF, there are also a few more subtle differences between CLF and CL file usage. This section is intended for CL developers accustomed to CL file processing and provides an overview of some of these differences.

### ***CL Variable Types***

#### **Binary Fields**

Prior to V5R3 of the operating system CL did not have direct support for the declaring of integer (or binary) variables. When the Declare File (DCLF) command encountered a binary field within a database file the CL compiler declared a packed decimal (\*Dec) variable. By default CL continues to use the \*Dec data type though this can be overridden with the Declare binary fields (DCLBINFLD) parameter of the DCLF command when the database field is defined with a precision of 0 decimal positions. As CLF requires that your system be at V5R4 (or higher) of the operating system, and integer fields perform better than packed decimal fields, CLF defaults to declaring binary fields as type integer (\*Int) if the binary field is defined with zero decimal positions. This default can be overridden when using CLF precompiler support by explicitly declaring (DCL) the CL variable as type \*Dec before the FILE command is encountered in the CL source program. The FILE command will also need to specify FLDSTG(\*AUTO).

CL, when encountering binary fields with a non-zero number of decimal positions, declares a \*Dec field of the appropriate length and precision. CLF, when using the precompiler and FILE FLDSTG(\*AUTO) will also declare a \*Dec field of the appropriate length and precision. If the FILE command specifies FLDSTG(\*DEFINED), or if the GENFFDCLF command is used, CLF will declare the CL variable as an integer field with zero decimal positions. A warning message (VC28028) will be found in the precompiler listing and the CL developer is responsible for adjusting the value of the integer field to the correct decimal precision. If the CL application will be working with binary fields of non-zero precision use of the CLF precompiler and FILE FLDSTG(\*AUTO) should be strongly considered over GENFFDCLF. Alternatively, a logical file can be created mapping the non-zero precision binary fields to an appropriate packed decimal field. This logical file would then be used by the CLF application (whether it be developed using the precompiler, the generation tools, or the base run-time support) as is.

#### **Variable-length fields**

CL, when encountering variable-length fields, declares a single field of type \*CHAR with a length equal to 2 bytes plus the maximum field length. The FILE command defaults to

declaring a field of type \*CHAR with a length equal to the maximum field length. This allows you to work directly with the contents of the variable-length field. The actual length of the field can be retrieved using the Retrieve Variable Field Length (RTVVFLCLF) command and set using the Set Variable Field Length (SETVFLCLF) command. This default can be overridden using the Variable field length support (VFLSPT) parameter of the FILE command. Specifying VFLSPT(\*FIXED) is compatible with the DCLF support of variable-length fields.

### ***Opening a Database File***

Conventional CL file support does not utilize an explicit open of files. Files are implicitly opened when they are first used by a command such as Receive File (RCVF). With CLF files are explicitly opened using the OPEN command.

### ***End-of-File Processing***

Conventional CL file support does not allow further file access after end-of-file has been reached. You must either:

- close the file using the CLOSE command and then implicitly re-open the file (available with V6R1 of the i operating system)
- or write multiple CL programs where a given CL program can be ended and restarted in order to re-read records from the file

CLF treats end-of-file as a temporary condition. After reaching end-of-file you can continue to access records in the file.

### ***Closing a Database File***

Conventional CL file support implicitly closes open files when the CL program ends. With CLF files are explicitly closed using the CLOSE command. Failure to close files prior to returning from a CLF application program can leave file resources allocated. In the case of an unmonitored exception, causing the CLF application program to end, file resources allocated by the application are freed.

## DATABASE FILE CAPABILITY LANGUAGE COMPARISON

This section looks at the database file capabilities provided by the various development languages available to you. The RPG, COBOL, and C capabilities are based on the ILE language operations available. In the case of RPG the comparison is with free-form syntax.

A check mark (✓) indicates that the language has direct support for the capability. The lack of a check mark does not necessarily mean that you as a developer cannot provide the capability within the given programming language. The lack of a check mark indicates that the developer may need to use multiple statements and/or use system APIs in order to provide the capability.

The CLF column assumes the use of a CLF precompiler when working with data attributes such as variable-length fields, null capable fields, etc.

Capability	CLF	CL	RPG	COBOL	C
Open file for input	✓	✓	✓	✓	✓
Multiple record formats	✓		✓	✓	✓
Read next record sequentially by active access path	✓	✓	✓	✓	✓
Read next record with key equal to current record value	✓				✓
Read next record with key equal to explicit value	✓		✓		
Read next record with key unique from current record value	✓				✓
Read next record with key unique from explicit value	✓				
Read previous record sequentially by active access path	✓		✓	✓	✓

Capability	CLF	CL	RPG	COBOL	C
Read previous record with key equal to current record value	✓				✓
Read previous record with key equal to explicit value	✓		✓		
Read previous record with key unique from current record value	✓				✓
Read previous record with key unique from explicit value	✓		✓		
Re-read same record	✓				✓
Randomly read first record by active access path	✓			✓	✓
Randomly read last record by access path	✓			✓	✓
Randomly read record with key equal to	✓		✓	✓	✓
Randomly read record with key less than or equal to	✓				✓
Randomly read record with key less than	✓				✓
Randomly read record with key greater than or equal to	✓				✓
Randomly read record with key greater than	✓				✓
Dynamic change of read operators and direction	✓				
Composite key support	✓		✓	✓	✓
Partial composite key support	✓		✓	✓	✓

Capability	CLF	CL	RPG	COBOL	C
Position to first record by active access path	✓		✓		✓
Position to last record by active access path	✓		✓		✓
Position to record with key equal to	✓		✓	✓	✓
Position to record with key less than or equal to	✓				✓
Position to record with key less than	✓				✓
Position to record with key greater than or equal to	✓		✓	✓	✓
Position to record with key greater than	✓		✓	✓	✓
Position to next record with key unique from current record value	✓				✓
Position to next record with key unique from explicit value	✓		✓		✓
Position to prior record with key unique from current record value	✓				✓
Position to prior record with key unique from explicit value	✓				
Randomly read file by RRN	✓		✓	✓	✓
Open file for output	✓		✓	✓	✓
Write new records	✓		✓	✓	✓

Capability	CLF	CL	RPG	COBOL	C
Ignore File REUSEDLT attribute	✓				
Explicitly write over deleted records	✓		✓	✓	✓
Open file for input/output	✓		✓	✓	✓
Update records	✓		✓	✓	✓
Release locked record	✓		✓		✓
Force buffered data to permanent storage	✓		✓		✓
Commit	✓		✓	✓	✓
Rollback	✓		✓	✓	✓
Retrieve file feedback information	✓		✓	✓	✓
Null capable field support	✓	1	✓	✓	✓
Variable-length field support	✓	1	✓	✓	1
Date, Time, Timestamp support	✓ <sup>2</sup>	1	✓	✓	1
Open file containing UDTs, Row IDs, data links, LOBs	✓				
Close file	✓	✓	✓	✓	✓

---

<sup>1</sup> Toleration for the attribute is provided, but no direct support

<sup>2</sup> Duration support is available with PowerCL: Control Language Tools (CLT). CLF provides limited support for these datatypes.

## 9. Using Display Files

You can access a display file from your program by opening the file using the Open File for Processing (OPEN) command. On the OPEN command you provide a File identifier (FILEID) that is used by other CLF commands when referencing the file. The FILE keyword of the OPEN command allows you to optionally specify the display file and library to associate with the FILEID.

Display files can be either externally-described or program-described. If externally described, you can use a separate indicator area with the SEPINDARA keyword of the OPEN command. When developing CLF applications without the use of the precompiler you will find that using a separate indicator area can simplify the application.

The RPG-like commands that are supported with display files are:

- CHAIN – Random read of a subfile record
- CLOSE – Close a file
- EXFMT – Write to and then read from a file
- OPEN – Open a file
- READ – Read the next record within a file
- READC – Read the next changed record within a subfile
- UPDATE – Update a record within a subfile
- WRITE – Write a new record to a file

The CLF commands that are supported with display files are:

- ACQDEVCLF – Acquire a device for a file
- CLOFCLF – Close a file
- OPNFCLF – Open a file
- READRCDCLF – Read a record from a file
- RLSDEVCLF – Release a device from a file

- RTVFINFCLF – Retrieve information about a file
- SETDEVCLF – Set the current device for a file
- UPDRCDCLF – Update a record within a file
- WRKOPNFCLF – Work with open files
- WRTRCDCLF – Write a new record to a file
- WRTREADCLF – Write to and then read from a file
- WRTRRNCLF – Write a new record to a subfile by relative record number

The following IBM provided commands can be used with CLF display files:

- OVRDSPF – Override file used or file parameters

The following CLF commands can be used to extract file, record format, and field information from a display file:

- FILE – Extract file, record format, and field information during the precompile step of program creation
- GENFFDCLF -- Generate file, record format, and field information to be stored in a source file member
- DCLFCLF – Extract file, record format, and field information during the precompile step of program creation

Refer to the online help text provided with all CLF commands for information related to a given command's usage and considerations.

## WORKING WITH SUBFILES

The DDS for a subfile consists of two record formats: a subfile-record format and a subfile control-record format. The subfile-record format contains the field information that you want to treat as a list of records. To write a subfile-record you use the WRITE command with the RRN keyword. The Relative record number keyword (RRN) identifies the record of the subfile you want to write. When you write to a subfile-record format the information does not get sent to the display immediately. Your program writes the appropriate number of subfile-record format entries and then writes the subfile control-record format. The writing of the subfile control-record format controls whether or not the subfile-record format information is actually sent to the display device.

To read subfile-record formats after the user has entered or changed data within the subfile you can use the CHAIN command. If you are only interested in the subfile-record formats that the user has actually modified you can use the READC command when reading the subfile. To determine the relative record number of the changed subfile-record format within the subfile you can use the Retrieve File Information using CLF (RTVFINFCLF) command with the Current relative record number keyword CURRRN.

A complete example of using a subfile (actually two subfiles) can be found in the online help for the READC command.

Another example can be found in Figure 9. 3. This sample provides a Work With capability for the PRTCTL file that you used back in chapter 4.

## WORK WITH PRINT CONTROL

In Chapter 4, Getting Started, you created commands to add records to the PRTCTL database, update records in the PRTCTL database, and delete records from the PRTCTL database. You will now see what is required to provide a Work With Print Control Users function so that users do not need to know of the commands you created earlier.

Figure 9. 1 shows the source for the command WRKPRTUSR, Figure 9. 2 the source for display file PRTCTLWRK, and Figure 9. 3 the source for program RPG\_WRKUSR, the CPP for the WRKPRTUSR command. The RPG\_WRKUSR program allows a user to add, update, copy, and delete entries in the PRTCTL database of Chapter 4 Getting Started. The copy function, as you will see, is implemented using the ADDPRTUSR command.

```
CMD          PROMPT('Work with Print Control Users')
PARM        KWD(USRPRF) TYPE(*GENERIC) LEN(10) DFT(*ALL) +
           SPCVAL((*ALL)) PROMPT('User profile')
```

Figure 9. 1 - The WRKPRTUSR command source

The WRKPRTUSR command defines one parameter – the user profile that you want to work with. The User profile name can be specified as a discrete name (BAKER), a generic name (B\*), or the special value \*ALL.

The source for WRKPRTUSR can be found in member WRKPRTUSR of source file VC2CLF/QCMDSRC. To create the command into QTEMP you can use the command

```
CRTCMD CMD(QTEMP/WRKPRTUSR) PGM(QTEMP/RPG_WRKUSR) +
      SRCFILE(VC2CLF/QCMDSRC)
```

```
CA03(03)
R SFLPRT
SFL
* Subfile listing users and reports
SFLOPT          1  I  7  3VALUES(' ' '1' '2' '3' '4')
N60              DSPATR(PC)
USRPRF          R    B  7  7REFFLD(USRPRF PRTCTL)
60              DSPATR(PR)
PRTF            R    B  7  21REFFLD(PRTF PRTCTL)
60              DSPATR(PR)
OUTQ            R    B  7  35REFFLD(OUTQ PRTCTL)
60              DSPATR(PR)
COPIES          R    B  7  49REFFLD(COPIES PRTCTL)
```

```

R SFLCTLPRT                SFLCTL(SFLPRT)
* Control record for subfile listing users
                                SFLSIZ(11) SFLPAG(10)
                                OVERLAY ERASEINP(*ALL)
21                                SFLDSPCTL SFLDSP
21                                SFLEND(*MORE)
N21                               SFLCLR
                                1 26'List of PRTCTL Users:'
                                COLOR(WHT)
                                USRPRF OUT    10    B 1 57COLOR(WHT)
                                3 2'Type options, press Enter'
                                COLOR(BLU)
                                4 4'1 - Add      '
                                COLOR(BLU)
                                4 17'2 - Change  '
                                COLOR(BLU)
                                4 30'3 - Copy    '
                                COLOR(BLU)
                                4 43'4 - Remove  '
                                COLOR(BLU)
                                6 2'Opt'
                                COLOR(WHT)
                                6 7'User      '
                                COLOR(WHT)
                                6 21'Report   '
                                COLOR(WHT)
                                6 35'Queue    '
                                COLOR(WHT)
                                6 49'Copies   '
                                COLOR(WHT)

R KEY
* Command key prompt
                                22 2'F3=Exit'

```

Figure 9. 2 - DDS source for the PRTCTLWRK display file

The PRTCTLWRK display file defines three record formats. Format SFLPRT is a subfile used to list the PRTCTL record entries that match the criteria specified with the USRPRF keyword of the WRKPRTUSR command. Format SFLCTLPRT is the subfile control record for subfile SFLPRT and defines the field USRPRF\_OUT. USRPRF\_OUT allows the user to view and change the search criteria from the original use of the WRKPRTUSR command. Format KEY is used to inform the user that command key 3 can be used to exit the application. Command key 3 is defined at the file level to turn on indicator 03 when F3 is pressed.

The source for display file PRTCTLWRK can be found in member PRTCTLWRK of source file VC2CLF/QDDSSRC. To create the display file into QTEMP you can use the command

```
CRTDSPF FILE (QTEMP/PRTCTLWRK) SRCFILE (VC2CLF/QDDSSRC)
```

```

/*****/
/* Declare the program parameters */
/* &USRPRF_IN - The user profile name to be used */
/* in loading the Work With subfile. Can be */
/* the special value *ALL or a generic. */
/*****/

Pgm          Parm(&UsrPrf_In)
Dcl          Var(&UsrPrf_In)  Type(*Char) Len(10)

/*****/
/* Declare working variables */
/* &USR_COUNT - used as relative record number */
/* when loading records to subfile */
/* &USRPRF_LEN - length of user profile field */
/* &SCAN_LOC - location of * in user profile */
/* name. Used for generic name */
/* support */
/* &CMP_LEN - length of user profile name to */
/* use for including in subfile */
/* &CMD - command to run when adding a user */
/* &CMD_LEN - maximum length of command to run */
/* &COPIES_CHR - &Copies in character form */
/* &PROMPT_KEY - command key used from command */
/* prompt panel */
/* &ONE - the number 1. Used in calls to */
/* the QCLSCAN API */
/*****/

Dcl          Var(&Usr_Count)  Type(*Dec)  Value(0)
Dcl          Var(&UsrPrf_Len) Type(*Dec)  Len(3 0) +
              Value(10)
Dcl          Var(&Scan_Loc)   Type(*Dec)  Len(3 0)
Dcl          Var(&Cmp_Len)    Type(*UInt)
Dcl          Var(&Cmd)        Type(*Char) Len(100)
Dcl          Var(&Cmd_Len)    Type(*Dec)  Len(15 5) +
              Value(100)
Dcl          Var(&Copies_Chr) Type(*Char) Len(3)
Dcl          Var(&Prompt_Key) Type(*Char) Len(3)
Dcl          Var(&One)        Type(*Dec)  Len(3 0) +
              Value(1)

/*****/
/* Declare files and indicators: */

```

```

/* PRTCTL - Print Control User file          */
/* PRTCTLWRK - Display file                 */
/* CLF command related indicators          */
/*****/

File      FileID(PRTCTL)
File      FileID(PRTCTLWRK)
Inds      CLFInd(*Yes)

/*****/
/* Open the files.                          */
/*****/

Open      PRTCTL AccMth(*Key)
Open      PRTCTLWRK Usage(*Both)

/*****/
/* Set the initial search value for subfile */
/*****/

ChgVar    Var(&UsrPrf_Out) Value(&UsrPrf_In)

/*****/
/* Loop on Work With panel until F3 is used */
/*****/

DoWhile   Cond(*Not &IN03)

    /*****/
    /* Clear the subfile                      */
    /*****/

    ChgVar Var(&IN21) Value('0')
    Write RcdFmt(SFLCTLPR)

    /*****/
    /* Write one blank record first. This record */
    /* allows input of UsrPrf, PrtF, OutQ and    */
    /* Copies as it is intended for Add operations.*/
    /*                                           */
    /* Then disable input for existing records by */
    /* setting indicator 60 (&IN60) to on.      */
    /*****/

    ChgVar Var(&UsrPrf)      Value(' ')
    ChgVar Var(&PrtF)        Value(' ')
    ChgVar Var(&OutQ)        Value(' ')
    ChgVar Var(&Copies)      Value(0)
    ChgVar Var(&Usr_Count)   Value(1)

    ChgVar Var(&IN60)        Value('0')
    Write RcdFmt(SFLPR) RRN(&Usr_Count)
    ChgVar Var(&IN60)        Value('1')

```

```

/*****/
/* Load list of users into subfile */
/*****/

/*****/
/* First determine the user to start with */
/*****/

/*****/
/* If the end user blanked out the profile */
/* name on the screen we will use *ALL */
/*****/

If Cond(&UsrPrf_Out = ' ') Then( +
    ChgVar Var(&UsrPrf_Out) Value(*ALL))

/*****/
/* Find if an '*' is in the profile name we */
/* are to show */
/*****/

Call Pgm(QCLSCAN) Parm(&UsrPrf_Out &UsrPrf_Len +
    &One '*' &One '0' '0' ' ' ' ' +
    &Scan_Loc)

Select
    When Cond(&Scan_Loc = 0) Then(Do)

        /*****/
        /* No '*' found. Load only those names */
        /* that match for all ten characters */
        /*****/

        ChgVar Var(&Cmp_Len) Value(10)
        SetLL &UsrPrf_Out PRTCTL
        EndDo

    When Cond(&Scan_Loc = 1) Then(Do)

        /*****/
        /* '*' in position 1 means *ALL was */
        /* entered. Load all records with no */
        /* comparison for matching names */
        /*****/

        ChgVar Var(&Cmp_Len) Value(0)
        SetLL *Loyal PRTCTL
        EndDo

    When Cond(&Scan_Loc > 1) Then(Do)

        /*****/
        /* '*' found. Load records only if */
        /* characters before the '*' match */
        /*****/

```

```

        ChgVar Var(&Cmp_Len) Value(&Scan_Loc - 1)
        SetLL %SST(&UsrPrf_Out 1 &Cmp_Len) PRTCTL
        EndDo

    Otherwise Cmd(Do)

        /*****/
        /* Unexpected error found. Send      */
        /* message and exit the program      */
        /*****/

        SndPgmMsg Msg('Unexpected error found')
        ChgVar Var(&IN03) Value('1')
        EndDo
    EndSelect

If Cond(&IN03) Then(Leave)

/*****/
/* Now load the records by reading the next */
/* record in PRTCTL. The correct position in */
/* PRTCTL was set in the previous SELECT group */
/*****/

Read PRTCTL EOF(&EOF)
DoWhile Cond(*Not &EOF)

/*****/
/* Does this record meet the search      */
/* criteria?                             */
/*****/

If Cond(&Cmp_Len > 0) Then( +
    If Cond(%SST(&UsrPrf 1 &Cmp_Len) *NE +
        %SST(&UsrPrf_Out 1 &Cmp_Len)) +
        Then(Do)

            /*****/
            /* Record does not match so */
            /* set &EOF to on and leave */
            /* the DOWHILE loop        */
            /*****/

            ChgVar Var(&EOF) Value('1')
            Leave
            EndDo

/*****/
/* Set subfile record number            */
/* Write user record                    */
/* Get next user record                  */
/*****/

```

```

      ChgVar Var(&Usr_Count) Value(&Usr_Count + 1)
(C)   Write RcdFmt(SFLPRT) RRN(&Usr_Count)
      Read PRTCTL EOF(&EOF)
      EndDo

      /*****
      /* When all of the records have been loaded: */
      /* Display the command key prompt and      */
      /* display the subfile and subfile control  */
      /* record (conditioned by indicator 21)     */
      /*****/

(D)   Write RcdFmt(KEY)
      ChgVar Var(&IN21) Value('1')
      ExFmt RcdFmt(SFLCTLPRT)

      /*****
      /* Test if command key 3 was used and leave */
      /* the DoWhile loop if it was              */
      /*****/

      If Cond(&IN03) Then(Leave)

      /*****
      /* Read subfile records to see what function */
      /* is requested                             */
      /*****/

      Readc RcdFmt(SFLPRT) EOF(&EOF)

      DoWhile Cond(*Not &EOF)

      /*****
      /* Perform the function                      */
      /*****/

      Select
        When Cond(&SFLOpt = '1') Then(Do)

          /*****
          /* For an add determine what           */
          /* values the user might have         */
          /* already entered, create an        */
          /* appropriate command string,       */
          /* and run the command               */
          /*****/

          ChgVar Var(&Cmd) Value('?AddPrtUsr')
          If Cond(&UsrPrf *NE ' ') Then( +
            ChgVar Var(&Cmd) Value( +
              &Cmd *BCat '??UsrPrf(' +
                *TCat &UsrPrf *TCat ' '))
          If Cond(&PrtF *NE ' ') Then( +
            ChgVar Var(&Cmd) Value( +

```

```

        &Cmd *BCat '??PrtF(' +
        *TCat &PrtF *TCat ')')
If Cond(&OutQ *NE ' ') Then( +
    ChgVar Var(&Cmd) Value( +
        &Cmd *BCat '??OutQ(' +
        *TCat &OutQ *TCat ')')
If Cond(&Copies *NE 0) Then(Do)
    ChgVar Var(&Copies_Chr) +
    Value(&Copies)
    ChgVar Var(&Cmd) Value( +
        &Cmd *BCat '??Copies(' +
        *TCat &Copies_Chr *TCat ')')
EndDo
Call Pgm(QCMDEXC) Parm(&Cmd &Cmd_Len)
MonMsg MsgID(CPF6801) Exec(Do)

```

```

/*****/
/* F3 or F12 was used on */
/* prompt panel */
/*****/

```

```

CallSubr Subr(GetMsg)
EndDo

```

EndDo

When Cond(&SFLOpt = '2') Then(Do)

```

/*****/
/* For a change allow the user to */
/* only change the output queue */
/* and the number of copies */
/*****/

```

```

ChgPrtUsr ?*UsrPrf(&UsrPrf) +
    ?*PrtF(&PrtF) ??OutQ(&OutQ) +
    ??Copies(&Copies)
MonMsg MsgID(CPF6801) Exec(Do)

```

```

/*****/
/* F3 or F12 was used on */
/* prompt panel */
/*****/

```

```

CallSubr Subr(GetMsg)
EndDo

```

EndDo

When Cond(&SFLOpt = '3') Then(Do)

```

/*****/
/* For a copy have the user supply */
/* the user profile and printer */
/* file names */
/*****/

```

```

?AddPrtUsr OutQ(&OutQ) Copies(&Copies)
MonMsg MsgID(CPF6801) Exec(Do)

    /*****/
    /* F3 or F12 was used on */
    /* prompt panel */
    /*****/

    CallSubr Subr(GetMsg)
    EndDo
EndDo

When Cond(&SFLOpt = '4') Then(Do)

    /*****/
    /* For a remove the user cannot */
    /* change anything. Just confirm */
    /* by pressing Enter */
    /*****/

    RmvPrtUsr ?*UsrPrf(&UsrPrf) +
              ?*PrtF(&PrtF)
    MonMsg MsgID(CPF6801) Exec(Do)

        /*****/
        /* F3 or F12 was used on */
        /* prompt panel */
        /*****/

        CallSubr Subr(GetMsg)
        EndDo
    EndDo

EndSelect

/*****/
/* Get the next requested function unless */
/* F3 was used. Just ignore F12 */
/*****/

If Cond(&IN03) Then(Leave)

Readc RcdFmt(SFLPRT) EOF(&EOF)

EndDo

EndDo

/*****/
/* Close the files as command key 3 was used */
/*****/

Close PRTCTL

```

```

Close      PRTCTLWRK

/*****
/* Start of subroutines
/*****

Subr      Subr (GetMsg)
RcvMsg   MsgQ (*PgmQ) MsgType (*Last) +
         MsgDta (&Prompt_Key)
If       Cond (&Prompt_Key = F3) Then (Do)

         /*****
         /* If F3 on command prompt then exit
         /* program. If F12 just ignore as
         /* command did not run
         /*****
         ChgVar Var (&IN03) Value ('1')
         ChgVar Var (&EOF) Value ('1')
         EndDo

EndSubr

EndPgm

```

Figure 9. 3 - The RPG\_WRKUSR program

The RPG\_WRKUSR program provides a reasonable Work With capability. There are a few items missing – option 1 (Add) should only apply when selected on the first blank subfile record, options 2, 3, and 4 should only apply for subfile records two and greater, etc, but these are not CLF or file related features. The example program does successfully demonstrate several features of CLF file support related to both database and display files.

After declaring the files that are used and opening them, the program moves the value of the passed parameter, &UsrPrf\_In, to the variable &UsrPrf\_Out. This variable is used for both input and output with the SFLCTLPRT subfile control record format. It's used to display to the user what selection criteria is in effect for the PRTCTL records that are being shown. This field is also defined as an input field so that the user can change the selection criteria without needing to exit the application.

After setting the &UsrPrf\_Out variable the program enters a DOWHILE loop that is conditioned by indicator variable &IN03. Indicator &IN03 is set on by the user using command key 3 on any record format of the display file. So as long as command key 3 is not pressed, the application will continue to display PRTCTL records.

Within the DOWHILE loop the program clears the subfile using the WRITE command at (A). This is in preparation of loading the appropriate PRTCTL records given the current value of &UsrPrf\_Out. Indicator &IN21, when off, runs the DDS subfile clear (SFLCLR) function.

RPG\_WRKUSR then writes one blank subfile record at (B) using the WRITE command. The Relative record number (RRN) keyword is used to identify the record that is to be written. At this point in the program the variable &Usr\_Count is set to the value of 1 so the first record of the subfile is written. This blank record is written so that the user can add a new PRTCTL entry using option 1 from the Work With panel.

Having written the blank subfile record entry RPG\_WRKUSR then determines what records of PRTCTL the user wants to have displayed. To do this PRTCTL uses the QCLSCAN API to scan the search criteria (&UsrPrf\_Out) to determine if an asterisk (\*) is found within the character string. If no asterisk is found QCLSCAN will return the value 0 in CL variable &Scan\_Loc (Scan location). If an asterisk is found in the CL variable &UsrPrf\_Out QCLSCAN will return the location of the asterisk in the CL variable &Scan\_Loc.

With this background on the QCLSCAN API the program enters a SELECT group.

If no asterisk is found in &UsrPrf\_Out the program sets the variable &Cmp\_Len (Comparison length) to a value of 10. This value is used later in the program to test that all ten characters of a user profile name read from the PRTCTL file match the full ten characters of &UsrPrf\_Out. For a full compare of the profile name the program runs the SETLL command to position the PRTCTL file to the first record with a key value that is greater than or equal to the value of &UsrPrf\_Out.

If an asterisk is found in the first character position of &UsrPrf\_Out RPG\_WRKUSR sets the variable &Cmp\_Len to a value of 0. This indicates that the special value \*ALL was entered and is used later in the program to cause no character comparison to be done between a user profile name read from the PRTCTL file and the value of &UsrPrf\_Out. The program in this situation runs the SETLL command with KEYLIST set to \*LOVAL. This positions the PRTCTL file to the first record in the file.

If an asterisk is found in any other character position of &UsrPrf\_Out RPG\_WRKUSR sets the variable &Cmp\_Len to the value of the position (&Scan\_Loc) less 1. This value is used later in the program to test that all characters, up to but not including the asterisk, of the user profile name read from the PRTCTL database file match the same character positions of &UsrPrf\_Out. The program for this situation runs the SETLL command with a KEYLIST argument of the characters in &UsrPrf\_Out up to the asterisk. This will position the PRTCTL file to the first record with a key value that is greater than or equal to the substringed value of &UsrPrf\_Out.

If the value of &Scan\_Loc is not within the range of 0 to 10 then an error is reported to the user and the program exits.

As the PRTCTL file is now positioned to the first record to be processed (due to the previous SELECT group logic), RPG\_WRKUSR now reads a record using the READ command with the End of File (EOF) keyword specifying that indicator &EOF should be

set to on when end of file is encountered. If &EOF is off upon return from the READ command the program falls into a DOWHILE loop that exits when end-of-file is detected. End of file in this situation does not necessarily mean physical end of file. Only that there are no more records that meet the &UsrPrf\_Out criteria.

For each record read the program uses the value of variable &Cmp\_Len to verify that the user profile name read (&UsrPrf) matches the necessary characters of &UsrPrf\_Out. If they do not match indicator &EOF is set to on and the DOWHILE loop ends. If the necessary characters do match the record is added to the subfile using the WRITE command at (C). Prior to writing each record the program increments the CL variable &Usr\_Count by 1 so that the new subfile record is written as the next entry in the subfile. The program then reads the next record of PRTCTL, again using the READ command with EOF(&EOF), and re-runs the DOWHILE loop.

Once all records that qualify for display have been written to the subfile RPG\_WRKUSR uses the WRITE command at (D) to write the command key prompts (record format KEY) to the display and the EXFMT command to write both the subfile and subfile control record. Indicator &IN21, when on, causes both the subfile and subfile control record to be displayed.

After the user has completed working with the list of PRTCTL entries RPG\_WRKUSR checks to see if command key 3 (represented by variable &IN03) was used. If so the program exits. If not, the program uses the READC command with keyword EOF(&EOF). Using the READC command indicates that only subfile records where the user typed in an option (that is, they changed the subfile record) should be read. When all changed subfile entries have been read, the indicator &EOF is to be set on.

For each subfile record read the program then processes the request in a SELECT group. The actual processing is not explained in any detail as there are no CLF commands used (only the commands you created earlier), but in brief RPG\_WRKUSR prompts and then runs the appropriate ADDPRTUSR, CHGPRTUSR, or RMVPRTUSR command from Chapter 4. The program, after each command is prompted, also checks to see if command key 3 was used to exit the command prompting. If so RPG\_WRKUSR exits. If command key 3 was not used the program reads the next changed record using the READC command.

After all work with requests have been processed (that is the variable &EOF is set to on) RPG\_WRKUSR re-enters the DOWHILE loop creating an updated subfile containing any updates that might have taken place.

The source for RPG\_WRKUSR can be found in member RPG\_WRKUSR of source file VC2CLF/VC2CLSRC. To create the program into QTEMP (assuming you have previously created the display file PRTCTLWRK of Figure 9. 2) you can use the command

```
CRTBNDCLF PGM(QTEMP/RPG_WRKUSR) SRCFILE(VC2CLF/VC2CLSRC)
```

You could have created this “Work with” function in the past using a language such as RPG, COBOL, or C, and perhaps you have written such a program. In that case you are already aware that making use of the commands ADDPRTUSR, CHGPRTUSR, and RMVPRTUSR, as was done with RPG\_WRKUSR, is not as straight forward as simply doing everything in a CL application program.

From a RPG, COBOL, or C program you need to use system APIs such as QCMDEXC, QCAPCMD, or system() in order to run the appropriate CL command. If you want to run a CL command that returns information to an RPG, COBOL, or C application program variable you have to write a CL program to call. If you want your non-CL program to be aware of error situations you need to monitor for program and/or command failures (which can, at times, be challenging when the monitoring is being done by your non-CL language program), etc. All of this additional work can be eliminated by using the file capabilities of CLF. You can simply provide the “Work with” function in the language most suitable for the task – namely CL.

## USING THE PRECOMPILER AND CL FILE SUPPORT IN THE SAME APPLICATION

You may have existing CL applications using traditional display file support. The following example demonstrates the ease in which you can add CLF database function while maintaining your existing CL file support. You can of course also replace the CL SNDRCVF commands if you desire to, but that's your choice. With CLF you certainly don't have to if there is no business reason to do so.

The display file being used is the same display file used in Chapter 4 -Using a Display File.

```

/*****/
/* This program provides an inquiry function */
/* to the PRTCTL file */
/*****/

Pgm

/*****/
/* Declare the PRTCTL database file using CLF */
/*****/

File      FileID(PRTCTL)

/*****/
/* Declare the PRTCTLINQ display file using CL */
/*****/

DclF      File(PRTCTLINQ)

/*****/
/* Open PRTCTL */
/*****/

Open      FileID(PRTCTL) AccMth(*Key)

/*****/
/* So long as the user does not press CF03 */
/* prompt for the user and report name to show */
/*****/

DoUntil   Cond(&IN03)

          /*****/
          /* Display the PROMPT display */
          /*****/

```

```
SndRcvF RcdFmt (PROMPT)

If Cond(&IN03) Then(Leave)

Chain (&UsrPrf &PrtF) PRTCTL +
  RcdNotFnd(&IN50)

/*****/
/* If record found, display it */
/*****/

If Cond(*Not &IN50) Then( +
  SndRcvF RcdFmt (DISPLAY))

EndDo

/*****/
/* Close our files and return when CF03 used */
/*****/

Close      OpnID(PRTCTL)

EndPgm
```

Figure 9. 4 - Using CL file support with CLF precompiler database support

For all practical purposes the program shown in Figure 9. 4 is the same as the RPG\_DSPUSR program of Figure 4. 12. In terms of function provided, readability of the program, and maintainability of the program they are equivalent. Though there is nothing new in terms of CLF file support usage shown in the program, the program does demonstrate how CLF function can be added to existing CL programs that may already have files declared and in use.

The source for the program can be found in member RPG\_DSPUS2 of source file VC2CLF/VC2CLSRC. To create the program into QTEMP you can use the command

```
CRTBNDCLF PGM(QTEMP/RPG_DSPUS2) SRCFILE(VC2CLF/VC2CLSRC)
```

## DISPLAY FILE CAPABILITY LANGUAGE COMPARISON

This section looks at the display file capabilities provided by the various development languages available to you. The RPG, COBOL, and C capabilities are based on the ILE language operations available.

A check mark (✓) indicates that the language has direct support for the capability. The lack of a check mark does not necessarily mean that you as a developer cannot provide the capability within a given programming language. The lack of a check mark indicates that the developer may need to use multiple statements and/or use system APIs in order to provide the capability.

Capability	CLF	CL	RPG	COBOL	C
Open file	✓	✓	✓	✓	✓
Multiple record formats	✓	✓	✓	✓	✓
Separate indicator area	✓	✓	✓	✓	✓
Subfile support	✓		✓	✓	✓
Multiple device support	✓	✓	✓	✓	✓
Retrieve file feedback information	✓		✓	✓	✓

# 10. Using Printer Files

You can access a printer file from your program by opening the file using the Open File for Processing (OPEN) command. On the OPEN command you provide a File identifier (FILEID) that is used by other CLF commands when referencing the file. The FILE keyword of the OPEN command allows you to optionally specify the printer file and library to associate with the FILEID. The USAGE keyword of the OPEN command must specify \*OUTPUT when using printer files. You can also use the SPLFNAME and SPLFUSRDTA keywords of the OPEN command to specify a spool file name and spool file user data to aid in identifying the spooled report. Additional attributes of the spool file can be set using the Override Printer File (OVRPRTF) command prior to opening the printer file.

Printer files can be either externally-described or program-described. If externally described, you can optionally use a separate indicator area with the SEPINDARA keyword of the OPEN command.

The RPG-like commands that are supported with printer files are:

- CLOSE – Close a file
- OPEN – Open a file
- WRITE – Write a new record to a file

The CLF commands that are supported with printer files are:

- CLOFCLF – Close a file
- OPNFCLF – Open a file
- RTVFINFCLF – Retrieve information about a file
- WRKOPNFCLF – Work with open files
- WRTRCDCLF – Write a new record to a file

The following IBM provided commands can be used with CLF printer files:

- OVRPRTF – Override file used or file parameters

The following CLF commands can be used to extract file, record format, and field information from a printer file:

- FILE – Extract file, record format, and field information during the precompile step of program creation
- DCLFCLF – Extract file, record format, and field information during the precompile step of program creation
- GENFFDCLF -- Generate file, record format, and field information to be stored in a source file member

Refer to the online help text provided with all CLF commands for information related to a given command's usage and considerations.

Several example programs are provided in this chapter. These examples all create (essentially) the same report. The reports do differ in their heading in that the heading indicates if the report was created using an external printer file (\*EXTERNAL), a program-described \*FCFC printer file (\*FCFC), or a program-described \*MACHINE printer file (\*MACHINE). The report generated using an externally described printer file, as viewed from iSeries Navigator, is shown in Figure 10.1.

\*EXTERNAL

## Department Listing

Department: Corporate Management                      Managed by: Rebecca

<u>Employee:</u>	<u>Ext:</u>
Amie	1759
Bruce	4178
Mindy	6896

Department: Product Support                              Managed by: Amie

<u>Employee:</u>	<u>Ext:</u>
Kiernan	1759

Department: Product Development                      Managed by: Bruce

<u>Employee:</u>	<u>Ext:</u>
Dovid	8266
Hadleigh	8266
Matthew	8254
Moshe	8266
Richard	9093

Department: Administrative Support                      Managed by: Mindy

<u>Employee:</u>	<u>Ext:</u>
No active employees	

Department: Product Marketing                              Managed by: Richard

<u>Employee:</u>	<u>Ext:</u>
Betty	6736
Clayton	6736

Figure 10. 1- Sample report created by several sample programs

## EXTERNAL PRINTER FILES

Externally described printer files provide an easy and flexible way to generate reports from a CLF application. Complete examples of using an external print file can be found below. Figure 10.2 shows how to use an external printer file with the precompiler. Additional examples are provided in the online help for the OPEN command.

### *Example using External Print File and the CLF Precompiler*

```

/*****
/* This program creates a printed report      */
/* listing employees in first name sequence  */
/* by department.                            */
*****/

Pgm

/*****
/* Declare the files used and the CLF related */
/* indicators:                               */
/* VC2DPT   - Department master              */
/* VC2EMP   - Employee master - used to get  */
/*           department manager name         */
/* VC2EMPDPT - Employees keyed by first name */
/*           within department               */
/* VC2PRTEXT - Printer file                  */
*****/

File      VC2DPT
File      VC2EMP
File      VC2EMPDPT
File      VC2PRTEXT
Inds      CLFInd(*Yes)

/*****
/* Open the database files for keyed access  */
/* Open the printer file for output only     */
*****/

Open      VC2DPT  AccMth(*Key)
Open      VC2EMP  AccMth(*Key)
Open      VC2EMPDPT AccMth(*Key)
Open      VC2PRTEXT Usage(*Output)

/*****
/* Print title of report                      */
*****/

Write     RcdFmt(Title)

```

```

/*****/

/* Get first department record */
/*****/

Read      VC2DPT EOF(&EOF)

/*****/
/* Process all departments.  The end of this */
/* processing is conditioned by end-of-file */
/* on the department master VC2DPT */
/*****/

DoWhile   Cond(*Not &EOF)

          /*****/
          /* Get department managers name. */
          /* If record not found use blanks. */
          /*****/

          Chain &DptMgr VC2EMP RcdNotFnd(&RNF)

          If Cond(&RNF) Then( +
            ChgVar Var(&EmpFName) Value(' '))

          /*****/
          /* Print department name and manager*/
          /* and column headings */
          /*****/

          Write RcdFmt(Heading)

          /*****/
          /* Print all active employees by */
          /* positioning to first employee */
          /* in department */
          /*****/

          SetLL &DptID VC2EMPDPT
          ReadE &DptID VC2EMPDPT EOF(&EOF)

          /*****/
          /* If no employee record found: */
          /* Print 'No active employees' */
          /* Read next department record */
          /* Re-run the DoWhile loop */
          /*****/

          If Cond(&EOF) Then(Do)
            Write RcdFmt(NoEmpl)
            Read VC2DPT EOF(&EOF)
            Iterate
          EndDo

```

```

/*****/

/* Since employees found, we will */
/* now process all employees in */
/* department printing all active */
/* employees (&EmpSts = 'I' means */
/* inactive). End of department */
/* employees is detected by read */
/* *NXTEQ returning end-of-file */
/*****/

DoWhile Cond(*Not &EOF)
  If Cond(&EmpSts *NE I) Then( +
    Write RcdFmt(Detail))

    /*****/
    /* Read next employee in department */
    /* and continue in the DoWhile loop */
    /*****/

    ReadE &DptID VC2EMPDPT EOF(&EOF)
    EndDo

    /*****/
    /* Read next department and */
    /* continue in the DoWhile loop */
    /*****/

    Read VC2DPT EOF(&EOF)
    EndDo

/*****/
/* All departments have been processed so */
/* close the files */
/*****/

Close      VC2DPT
Close      VC2EMP
Close      VC2EMPDPT
Close      VC2PRTEXT

EndPgm

```

Figure 10. 2 - Using the precompiler with an external printer file

The source for this program can be found in member RPG\_PRTEXT of source file VC2CLSRC in library VC2CLF.

Figure 10.3 shows the DDS source for the printer file VC2PRTEXT.

```

.....AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++
R TITLE                                     SPACEA(1)
                                           1 '*EXTERNAL'
                                           29 'Department Listing'

R HEADING                                     SPACEB(3)
                                           1 'Department:'
DPTNAME          30                        +1
                                           +1 'Managed by:'
EMPFNAME         40                        +1SPACEA(2)
                                           4 'Employee:'
                                           45 'Ext:'
                                           4 '_____ '
                                           24 '_____ '
                                           45 '_____ '
                                           SPACEA(1)

R DETAIL
EMPFNAME         40                        4
EMPEXT          4                          45SPACEA(1)

R NOEMPL
                                           1 'No active employees'
                                           SPACEA(1)

```

Figure 10. 3 - The DDS for external printer file VC2PRTEXT

The source for the external printer file can be found in member VC2PRTEXT of source file QDDSSRC in library VC2CLF.

The printer file VC2PRTEXT should currently exist in library VC2CLF so to compile the program RPG\_PRTEXT you can:

```

ADDLIBLE LIB(VC2CLF)
CRTBNDCLF PGM(QTEMP/RPG_PRTEXT) SRCFILE(VC2CLF/VC2CLSRC)

```

## PROGRAM-DESCRIBED PRINTER FILES

While more tedious to use than external printer files you can write to program-described printer files by specifying printer control characters in the first character of each print line. There are three types of printer control characters:

- \*FCFC – the first character of every record contains an ANSI forms control character
- \*MACHINE – the first character of every record contains a machine code control character
- \*NONE – no print control characters are used and each printed line is single spaced

The type of printer control character to use is specified by the CTLCHAR keyword of the Create Printer File (CRTPRTF), Change Printer File (CHGPRTF), and Override Printer File (OVRPRTF) commands. The default is \*NONE.

As CL has never had the ability to write reports you should give strong consideration to using external printer files when adding report writing capabilities to your CL applications. The formatting of a report in CL variables is comparable to what is required to define an external printer file while the potential benefits of using an external printer file are significant. Three examples of these benefits to your productivity are:

- To change column headings, line spacing, or shift field positions within a print line with a program described printer file requires that you modify the source of the application program and recompile. To change column headings, line spacing, or shift field positions on a print line with externally described printer files in most cases only requires that you change the printer file and recreate it. For these common types of changes there is often no need to modify the logic of the application program when using external printer files.
- To merge your database information into print lines of a program described printer file requires that you use the CHGVAR command to construct the print line prior to writing the print line using the WRITE command. To merge database information into print lines of an externally described printer file requires that you only write the print line using the WRITE command. The coding and testing of the CHGVAR commands disappear as they are no longer necessary.

- To add an existing database field to a print line of a program described printer file requires that you modify the application program and recompile. To add an existing database field to a print line of an externally described printer file, when using the precompiler, you often only need to add the field to the printer file, recompile the printer file, and recompile the application program. There is no need to change the logic of the application program if the necessary database field has already been read by the application when preparing the print line. This is same as adding an existing field to display file record format if you are familiar with the flexibility and productivity of externally described display files.

If you are not currently using external printer files now is the ideal time to start. All of these benefits, and more, mean greater productivity for you. Avoid if possible continuing to use program described printer files as you might be doing with current report writing applications written in languages such as RPG and COBOL when moving to CL-based report writing programs. A glance at the coding differences between the program of Figure 10. 2 and the following Figure 10. 4 should demonstrate the advantages to you.

### **\*FCFC Processing**

When using \*FCFC printer control characters the first character of each print line defines what action is to be taken *before* printing the associated line. For instance a value of ' ' (blank) in the first position indicates to space one line, a value of '0' to space two lines, and a value of '+' to suppress any spacing. In source member FCFCPRTCTL of source file VC2CLF/VC2CLSRC you will find a definition for the \*FCFC printer control characters. This definition can be included within your CL source program either by use of the INCLUDE command or using a source editor option such as SEUs Browse/Copy option.

#### **Example using \*FCFC Processing and the CLF Precompiler**

Figure 10.4 shows a program generating the report of Figure 10.1 using a program described printer file with \*FCFC processing.

```

/*****/
/* This program creates a printed report      */
/* listing employees in first name sequence  */
/* by department.                            */
/*                                           */
/* Several different styles are used in      */
/* generating the print lines used in the    */
/* report. This is done so that you can see  */
/* the various ways that can be used to     */
/* create a report.                          */

```

```
/* */
/* The variable &Title is used to format the */
```

```
/* report heading. The first character of */
/* &Title is reserved for the printer control */
/* character. This printer control character */
/* is later set using the CL substring builtin */
/* with CHGVAR. */
/* */
/* The variables &Heading1 and &Heading2 are */
/* used to format underlined column headings. */
/* As with &Title, the printer control */
/* character is set with CHGVAR and the */
/* substring builtin. */
/* */
/* The variable &PrtLine is defined as a */
/* structure 133 characters in length. */
/* */
/* The subfield &CtlChr is defined as the */
/* first character of &PrtLine and is used */
/* as the printer control character. This */
/* variable is set in the program directly */
/* by name -- the substring builtin is not */
/* used. */
/* */
/* The subfields &EmpFName and &EmpExt are */
/* used to define where the employees first */
/* name and phone extension are to be printed. */
/* These names match the field names found in */
/* the database files VC2EMP and VC2EMPDPPT. */
/* Because these fields are defined prior to */
/* the DCLFCLF command references to the */
/* VC2EMP and VC2EMPDPPT files the precompiler */
/* will use the explicit DCLs found in the */
/* &PrtLine structure. This re-use eliminates */
/* the need for the developer to move (CHGVAR) */
/* the database fields to the &PrtLine */
/* subfields. */
/* */
/* The subfield &PrtTxt is used to print */
/* department heading information. This */
/* demonstrates how CHGVAR can be used to */
/* create a print line using CL operations */
/* such as concatenation (*CAT, *BCAT, etc). */
/*****/
```

Pgm

```
*****/
/* Define various print lines for the report */
*****/

*****/
/* &Title - A Title for the report */
```

```
/******
```

```
Dcl      Var(&Title)      Type(*Char) Len(133) Value( +  
' *FCFC                      Department Listing')
```

```
/******
```

```
/* &Heading1 - Column headings for report      */  
/* &Heading2 - Underscores for column headings */  
/******
```

```
Dcl      Var(&Heading1) Type(*Char) Len(133) Value( +  
'      Employee:                      Ext:')
```

```
Dcl      Var(&Heading2) Type(*Char) Len(133) Value( +  
'      _____ ')
```

```
/******
```

```
/* &PrtLine - the combined printer control      */  
/*          character and printed text          */  
/* &CtlChr  - the printer control character      */  
/* &EmpFName- the employee first name           */  
/* &EmpExt  - the employee phone extension      */  
/* &PrtTxt  - the entire printed line           */  
/******
```

```
Dcl      Var(&PrtLine)  Type(*Char) Len(133)  
Dcl      Var(&CtlChr)   Type(*Char) Stg(*Defined) +  
                      Len(1)   DefVar(&PrtLine)  
Dcl      Var(&EmpFName) Type(*Char) Stg(*Defined) +  
                      Len(40)  DefVar(&PrtLine 5)  
Dcl      Var(&EmpExt)   Type(*Char) Stg(*Defined) +  
                      Len(4)   DefVar(&PrtLine 46)  
Dcl      Var(&PrtTxt)   Type(*Char) Stg(*Defined) +  
                      Len(132) DefVar(&PrtLine 2)
```

```
/******
```

```
/* Declare the files used and the CLF related */  
/* indicators:                                */  
/* VC2DPT   - Department master               */  
/* VC2EMP   - Employee master - used to get  */  
/*          department manager name          */  
/* VC2EMPDPT - Employees keyed by first name */  
/*          within department                */  
/*          */                                */  
/* The QSYSPRT printer file is also declared */  
/* so that the precompiler will recognize the */  
/* CLOSE command as applying to a CLF accessed */  
/* file (rather than a CL accessed file. If    */  
/* the program did not declare the QSYSPRT    */  
/* then the CLOFCLF command would be used     */  
/* rather than the CLOSE command.             */  
/******
```

```
File      VC2DPT
File      VC2EMP
File      VC2EMPDPT
```

```
File      QSYSPRT
Inds      CLFInd(*Yes)
```

```
/* *****
/* Include the CLF provided *FCFC printer      */
/* control values                               */
/* *****
```

```
Include   SrcMbr(FCFCPRTCTL) SrcFile(VC2CLF/VC2CLSRC)
```

```
/* *****
/* Override the QSYSPRT printer file to the    */
/* desired values and then open QSYSPRT.       */
/* LVLCHK(*NO) is specified as we did not      */
/* generate any file description for the file.  */
/* *****
```

```
OvrPrtF   File(QSYSPRT) PageSize(*N 133) +
           CtlChar(*FCFC)
OpnFCLF   QSYSPRT Usage(*Output) LvlChk(*No)
```

```
/* *****
/* Open the database files for keyed access    */
/* *****
```

```
Open      VC2DPT AccMth(*Key)
Open      VC2EMP AccMth(*Key)
Open      VC2EMPDPT AccMth(*Key)
```

```
/* *****
/* Print title of report                       */
/* *****
```

```
ChgVar Var(%sst(&Title 1 1)) Value(&Prt_Spc0)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Title)
```

```
/* *****
/* Get first department record                 */
/* *****
```

```
Read      VC2DPT EOF(&EOF)
```

```
/* *****
/* Process all departments. The end of this    */
/* processing is conditioned by end-of-file    */
/* on the department master VC2DPT            */
/* *****
```

```
DoWhile   Cond(*Not &EOF)
```

```

/*****/
/* Get department managers name. */
/* If record not found use blanks. */
/*****/

Chain &DptMgr VC2EMP RcdNotFnd(&RNF)

If Cond(&RNF) Then( +
    ChgVar Var(&EmpFName) Value(' '))

/*****/
/* Skip 3 lines */
/* Print department name and manager*/
/*****/

ChgVar Var(&CtlChr) Value(&Prt_Spc3)
ChgVar Var(&PrtTxt) Value('Department:' +
    *BCat &DptName *Cat ' Managed by:' +
    *BCat &EmpFName)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&PrtLine)

/*****/
/* Skip 2 lines */
/* Print column headings */
/* Print underscores for the column */
/* headings without skipping lines. */
/*****/

ChgVar Var(%sst(&Heading1 1 1)) +
    Value(&Prt_Spc2)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Heading1)

ChgVar Var(%sst(&Heading2 1 1)) +
    Value(&Prt_Spc0)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Heading2)

/*****/
/* Print all active employees by */
/* positioning to first employee */
/* in department */
/* */
/* First clear the print line and */
/* then position to department */
/* employees */
/*****/

ChgVar Var(&PrtLine) Value(' ')

SetLL &DptID VC2EMPDPT
ReadE &DptID VC2EMPDPT EOF(&EOF)

/*****/
/* If no employee record found: */
/* Print 'No active employees' */
/*****/

```

```

/* Read next department record      */
/* Re-run the DoWhile loop         */
/*****/

If Cond(&EOF) Then(Do)
  ChgVar Var(&CtlChr) Value(&Prt_Spc1)
  ChgVar Var(&PrtTxt) Value( +
    'No active employees')
  WrtRcdCLF FileID(QSYSPRT) +
    RcdBuf(&PrtLine)
  Read VC2DPT EOF(&EOF)
  Iterate
EndDo

/*****/
/* Since employee records were     */
/* found, setup Employee DoWhile   */
/* processing by setting all       */
/* employees to skip 1 line and    */
/* then pring                      */
/*****/

ChgVar Var(&CtlChr) Value(&Prt_Spc1)

/*****/
/* Now process all employees in    */
/* department printing all active  */
/* employees (&EmpSts = 'I' means  */
/* inactive). End of department    */
/* employees is detected by read   */
/* *NXTEQ returning end-of-file    */
/*****/

DoWhile Cond(*Not &EOF)
  If Cond(&EmpSts *NE I) Then(Do)
    WrtRcdCLF FileID(QSYSPRT) +
      RcdBuf(&PrtLine)
  EndDo

  /*****/
  /* Read next employee in department */
  /* and continue in the DoWhile loop */
  /*****/

  ReadE &DptID VC2EMPDPT EOF(&EOF)
  EndDo

  /*****/
  /* Read next department and       */
  /* continue in the DoWhile loop   */
  /*****/

  Read VC2DPT EOF(&EOF)
  EndDo

```

```

/*****/
/* All departments have been processed so      */
/* close the files and delete the QSYSPRT      */
/* override that was made earlier             */
/*****/

Close      VC2DPT
Close      VC2EMP
Close      VC2EMPDPT
Close      QSYSPRT
DltOvr     File(QSYSPRT)

EndPgm

```

Figure 10. 4 - Using \*FCFC processing and program-described printer files

While the RPG\_PRTFC program uses RPG-like commands for database file access the WRTRCDCLF command is used when writing to the program described printer file. The RPG-like WRITE command is intended to be used with files with a record format that includes at least one field and that is declared within the program by the FILE (or DCLFCLF) command. Device files (printer and display) that are not externally described have a record length of 0 bytes with no fields defined. It is this lack of defined fields that requires the use of the WRTRCDCLF command. Note that this consideration does not apply to database files. Database files always have a record length independent of whether or not the file is externally described.

The source for this program can be found in member RPG\_PRTFC of source file VC2CLF/VC2CLSRC. To compile RPG\_PRTFC into QTEMP you can use

```
CRTBNDCLF PGM(QTEMP/RPG_PRTFC) SRCFILE(VC2CLF/VC2CLSRC)
```

### **\*Machine Processing**

When using \*Machine printer control characters the first character of each print line defines what action is to be taken *after* printing the associated line. For instance a value of x'09' in the first position indicates to space one line, a value of x'11' to space two lines, and a value of x'01' to suppress any spacing. In source member MCHPRTCTL of source file VC2CLSRC in library VC2CLF you will find a definition for the \*Machine printer control characters. This definition can be included within your CL source program either by use of the INCLUDE command or using a source editor option such as SEUs Browse/Copy option.

When using \*Machine processing the device type (DEVTYPE) specified for the printer file must be \*LINE or \*AFPDSLIN.

## Example using \*Machine Processing and the CLF Precompiler

```
/******  
/* This program creates a printed report */  
/* listing employees in first name sequence */  
/* by department. */  
/* */  
/* Several different styles are used in */  
/* generating the print lines used in the */  
/* report. This is done so that you can see */  
/* the various ways that can be used to */  
/* create a report. */  
/* */  
/* The variable &Title is used to format the */  
/* report heading. The first character of */  
/* &Title is reserved for the printer control */  
/* character. This printer control character */  
/* is later set using the CL substring builtin */  
/* with CHGVAR. */  
/* */  
/* The variables &Heading1 and &Heading2 are */  
/* used to format underlined column headings. */  
/* As with &Title, the printer control */  
/* character is set with CHGVAR and the */  
/* substring builtin. */  
/* */  
/* The variable &PrtLine is defined as a */  
/* structure 133 characters in length. */  
/* */  
/* The subfield &CtlChr is defined as the */  
/* first character of &PrtLine and is used */  
/* as the printer control character. This */  
/* variable is set in the program directly */  
/* by name -- the substring builtin is not */  
/* used. */  
/* */  
/* The subfields &EmpFName and &EmpExt are */  
/* used to define where the employees first */  
/* name and phone extension are to be printed. */  
/* These names match the field names found in */  
/* the database files VC2EMP and VC2EMPDPT. */  
/* Because these fields are defined prior to */  
/* the FILE command references to the */  
/* VC2EMP and VC2EMPDPT files the precompiler */  
/* will use the explicit DCLs found in the */  
/* &PrtLine structure. This re-use eliminates */  
/* the need for the developer to move (CHGVAR) */  
/* the database fields to the &PrtLine */  
/* subfields. */  
/* */  
/* The subfield &PrtTxt is used to print */  
/* department heading information. This */
```

```

/* demonstrates how CHGVAR can be used to      */
/* create a print line using CL operations      */
/* such as concatenation (*CAT, *BCAT, etc).   */
/*****/

Pgm

/*****/
/* Define various print lines for the report  */
/*****/

/*****/
/* &Title   - A Title for the report          */
/*****/

Dcl      Var(&Title)      Type(*Char) Len(133) Value( +
' *MACHINE                Department Listing')

/*****/
/* &Heading1 - Column headings for report     */
/* &Heading2 - Underscores for column headings */
/*****/

Dcl      Var(&Heading1) Type(*Char) Len(133) Value( +
'      Employee:                Ext:')

Dcl      Var(&Heading2) Type(*Char) Len(133) Value( +
'      _____')

/*****/
/* &PrtLine - the combined printer control    */
/*           character and printed text      */
/* &CtlChr  - the printer control character  */
/* &EmpFName- the employee first name        */
/* &EmpExt  - the employee phone extension  */
/* &PrtTxt  - the entire printed line       */
/*****/

Dcl      Var(&PrtLine)   Type(*Char) Len(133)
Dcl      Var(&CtlChr)   Type(*Char) Stg(*Defined) +
                        Len(1)   DefVar(&PrtLine)
Dcl      Var(&EmpFName) Type(*Char) Stg(*Defined) +
                        Len(40)  DefVar(&PrtLine 5)
Dcl      Var(&EmpExt)   Type(*Char) Stg(*Defined) +
                        Len(4)   DefVar(&PrtLine 46)
Dcl      Var(&PrtTxt)   Type(*Char) Stg(*Defined) +
                        Len(132) DefVar(&PrtLine 2)

/*****/
/* Declare the files used and the CLF related */
/* indicators:                                */
/* VC2DPT   - Department master               */
/* VC2EMP   - Employee master - used to get  */
/*           department manager name         */

```

```

/* VC2EMPDPT - Employees keyed by first name */
/*          within department                */
/*                                          */
/* The QSYSPRT printer file is also declared */
/* so that the precompiler will recognize the */
/* CLOSE command as applying to a CLF accessed */
/* file (rather than a CL accessed file. If   */
/* the program did not declare the QSYSPRT   */
/* then the CLOFCLF command would be used    */
/* rather than the CLOSE command.           */
/*****/

File      FileID(VC2DPT)
File      FileID(VC2EMP)
File      FileID(VC2EMPDPT)
File      QSYSPRT
Inds      CLFInd(*Yes)

/*****/
/* Include the CLF provided *Machine printer */
/* control values                            */
/*****/

Include    SrcMbr(MCHPRTCTL) SrcFile(VC2CLF/VC2CLSRC)

/*****/
/* Override the QSYSPRT printer file to the */
/* desired values and then open QSYSPRT.    */
/* LVLCHK(*NO) is specified as we did not   */
/* generate any file description for the file.*/
/*****/

OvrPrtF   File(QSYSPRT) DevType(*Line) +
           PageSize(*N 133) CtlChar(*Machine)
Open      FileID(QSYSPRT) Usage(*Output) +
           LvlChk(*No)

/*****/
/* Open the database files for keyed access */
/*****/

Open      FileID(VC2DPT)    AccMth(*Key)
Open      FileID(VC2EMP)    AccMth(*Key)
Open      FileID(VC2EMPDPT) AccMth(*Key)

/*****/
/* Print title of report and then skip 1 line */
/*****/

ChgVar Var(%sst(&Title 1 1)) Value(&Prt_Spc1)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Title)

/*****/
/* Get first department record                */

```

```

/*****/

Read      FileID(VC2DPT) EOF(&EOF)

/*****/
/* Process all departments.  The end of this */
/* processing is conditioned by end-of-file */
/* on the department master VC2DPT          */
/*****/

DoWhile   Cond(*Not &EOF)

          /*****/
          /* Get department managers name.   */
          /* If record not found use blanks.  */
          /*****/

Chain &DptMgr VC2EMP RcdNotFnd(&RNF)

If Cond(&RNF) Then( +
    ChgVar Var(&EmpFName) Value(' '))

          /*****/
          /* Skip 3 lines                      */
          /* Print department name and manager*/
          /* Then skip two lines              */
          /*****/

ChgVar Var(&CtlChr) Value(&NoPrt_Spc3)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&PrtLine)

ChgVar Var(&CtlChr) Value(&Prt_Spc2)
ChgVar Var(&PrtTxt) Value('Department:' +
    *BCat &DptName *Cat ' Managed by:' +
    *BCat &EmpFName)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&PrtLine)

          /*****/
          /* Print column headings with no    */
          /* line skip.  Then print the      */
          /* underscores for the column     */
          /* headings and skip 1 line after. */
          /*****/

ChgVar Var(%sst(&Heading1 1 1)) +
    Value(&Prt_Spc0)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Heading1)

ChgVar Var(%sst(&Heading2 1 1)) +
    Value(&Prt_Spc1)
WrtRcdCLF FileID(QSYSPRT) RcdBuf(&Heading2)

          /*****/
          /* Print all active employees by   */

```

```

/* positioning to first employee */
/* in department */
/* */
/* First clear the print line and */
/* then position to department */
/* employees */
/*****/

ChgVar Var(&PrtLine) Value(' ')
SetLL &DptID VC2EMPDPT
ReadE &DptID VC2EMPDPT EOF(&EOF)

/*****/
/* If no employee record found: */
/* Print 'No active employees' */
/* Read next department record */
/* Re-run the DoWhile loop */
/*****/

If Cond(&EOF) Then(Do)
    ChgVar Var(&CtlChr) Value(&Prt_Spc1)
    ChgVar Var(&PrtTxt) Value( +
        'No active employees')
    WrtRcdCLF FileID(QSYSPRT) +
        RcdBuf(&PrtLine)
    Read VC2DPT EOF(&EOF)
    Iterate
EndDo

/*****/
/* Since employee records were */
/* found, setup Employee DoWhile */
/* processing by setting all */
/* employees to print and then skip */
/* one line */
/*****/

ChgVar Var(&CtlChr) Value(&Prt_Spc1)

/*****/
/* Now process all employees in */
/* department printing all active */
/* employees (&EmpSts = 'I' means */
/* inactive). End of department */
/* employees is detected by READE */
/* returning end-of-file */
/*****/

DoWhile Cond(*Not &EOF)
    If Cond(&EmpSts *NE I) Then(Do)
        WrtRcdCLF FileID(QSYSPRT) +
            RcdBuf(&PrtLine)
    EndDo

```

```

/*****/
/* Read next employee in department */
/* and continue in the DoWhile loop */
/*****/

ReadE &DptID VC2EMPDPT EOF(&EOF)
EndDo

/*****/
/* Read next department and          */
/* continue in the DoWhile loop      */
/*****/

Read VC2DPT EOF(&EOF)
EndDo

/*****/
/* All departments have been processed so */
/* close the files and delete the QSYSPRT */
/* override that was made earlier        */
/*****/

Close      VC2DPT
Close      VC2EMP
Close      VC2EMPDPT
Close      QSYSPRT
DltOvr     File(QSYSPRT)

EndPgm

```

Figure 10. 5 - Using \*MACHINE processing with the precompiler

While the RPG\_PRTMCH program uses RPG-like commands for database file access the WRTRCDCLF command is used when writing to the program described printer file. The RPG-like WRITE command is intended to be used with files with a record format that includes at least one field and that is declared within the program by the FILE (or DCLFCLF) command. Device files (printer and display) that are not externally described have a record length of 0 bytes with no fields defined. It is this lack of defined fields that requires the use of the WRTRCDCLF command. Note that this consideration does not apply to database files. Database files always have a record length independent of whether or not the file is externally described.

The source for this program can be found in member RPG\_PRTMCH of source file VC2CLSRC in library VC2CLF.

To compile RPG\_PRTMCH you need to use either the Create Bound CLF Program (CRTBNDCLF) command or the Create CLF Program (CRTCLFPGM) command. To create an ILE version of RPG\_PRTMCH into QTEMP you would use

```
CRTBNDCLF PGM(QTEMP/RPG_PRTMCH) SRCFILE(VC2CLF/VC2CLSRC)
```

## PRINTER FILE CAPABILITY LANGUAGE COMPARISON

This section looks at the printer file capabilities provided by the various development languages available to you. The RPG, COBOL, and C capabilities are based on the ILE language operations available.

A check mark (✓) indicates that the language has direct support for the capability. The lack of a check mark does not necessarily mean that you as a developer cannot provide the capability within a given programming language. The lack of a check mark indicates that the developer may need to use multiple statements and/or use system APIs in order to provide the capability.

Capability	CLF	CL	RPG	COBOL	C
Open file	✓		✓	✓	✓
Multiple record formats	✓		✓	✓	✓
Separate indicator area	✓		✓	✓	✓
Retrieve file feedback information	✓		✓	✓	✓
Close file	✓		✓	✓	✓

# Appendix A. ILE RPG and CLF

ILE RPG and CLF have many common capabilities in terms of their file handling. If you are familiar with ILE RPG support for files then this section may assist you in mapping RPG file specifications and operation codes to CLF commands and keywords.

## RPG AND CLF FILE SUPPORT AND DECLARATION

RPG Control and File Specifications	CLF Equivalent
ALWNULL keyword of H-spec	ALWNULL keyword of OPEN command
CVTOPT keyword of H-spec	Date, Time, Timestamp, and Graphic fields are defined as TYPE(*CHAR). The level of variable-length field (VarChar and VarGraphic) definition support is determined by the VFLSPT keyword of the FILE command
File name (positions 7 – 16 of F-spec)	FILEID keyword of OPEN command
File type (position 17 of F-spec)	USAGE keyword of OPEN command
File addition (position 20 of F-spec)	USAGE(*BOTH) or USAGE(*OUTPUT) of OPEN command
Program described file  File format (position 22 of F-spec)  Record length (positions 23 – 27 of F-spec)  Length of key (positions 29 – 33 of F-spec)	No direct one-to-one mapping, but:  If program described LVLCHK(*NO) of OPEN command  Record length is implied by size of BFR parameter of IO related CLF commands  Key length is implied by size of KEYLIST or KEYSTRUCT keywords of RDRCDCLF
Externally described file	FILE command
Record address type (position 34 of F-spec)	ACCMTH keyword of OPEN command
Device (positions 36 – 42 of F-spec)	Determined at run-time by FILE keyword of OPEN command

COMMIT keyword of F-spec	CMTCTL keyword of OPEN command
EXTFILE keyword of F-spec	FILE keyword of OPEN command
EXTMBR keyword of F-spec	MBR keyword of OPEN command
IGNORE keyword of F-spec	Not including record format with RCDFMT keyword of FILE command
INCLUDE keyword of F-spec	RCDFMT keyword of FILE command
INDDS keyword of F-spec	SEPINDARA and INDARA keywords of OPEN command Data structure can be defined with INDARA keyword of INDS command
INFDS keyword of F-spec	RTVFINFCLF command
PREFIX keyword of F-spec	NAMMDF keyword of FILE command
USROPN keyword of F-spec	All CLF files are USROPN

## RPG AND CLF FILE OPERATIONS MAPPING

RPG Operation	CLF Equivalent
ACQ	ACQDEVCLF
CHAIN with key	CHAIN with either KEYLIST or KEYSTRUCT keyword
CHAIN with RRN	CHAIN with RRN keyword
CLOSE	CLOSE
COMMIT	COMMIT
DELETE	DELETE
%EOF	EOF keyword of SETLL or SETGT
	EOF keyword of READ, READC, READE, READP, READPE
%ERROR	ERR keyword of CLF commands
EXCEPT	WRITE
EXFMT	EXFMT
FEOD	FRCDTACLF
%FOUND	RCDNOTFND keyword of CHAIN
KLIST/KFLD	SETLL or SETGT with KEYLIST or KEYSTRUCT keyword
	CHAIN, READE, READPE with KEYLIST or KEYSTRUCT keyword
%LEN	RTVVFLCLF to query length of variable-length field
	SETVFLCLF to set length of variable-length field
NEXT	SETDEVCLF
%NULLIND	RTVNULACLF to query null attribute setting
	CHGNULACLF to change null attribute setting

OPEN	OPEN
%OPEN	RTVFINFCLF with OPEN keyword
POST	RTVFINFCLF
READ	READ
READC	READC
READE	READE
READP	READP
READPE	READPE
REL	RLSDEVCLF
ROLBK	ROLLBACK
SETGT	SETGT
SETGT *LOVAL	SETGT *LOVAL
SETGT *HIVAL	SETGT *HIVAL
SETLL	SETLL
SETLL *LOVAL	SETLL *LOVAL
SETLL *HIVAL	SETLL *HIVAL
UNLOCK	UNLOCK
UPDATE	UPDATE
WRITE	WRITE

**RPG AND CLF FILE OPERATION CODE EXTENDER MAPPINGS**

RPG Extender	CLF Equivalent
--------------	----------------

(E)	ERR keyword of CLF commands
(N)	RCDLCK(*NO) of CLF read commands

#### RPG AND CLF MISCELLANEOUS MAPPINGS

RPG	CLF Equivalent
/Copy	INCLUDE

Notes:

RPG data structure based I/O file operations are similar to using FLDSTG(\*DEFINED) with the FILE command or using the GENFFDCLF command.

# Appendix B. VC2EMP and VC2DPT Sample Files

The CLF product provides two sample database files which are used extensively by the sample programs provided with the help text of CLF commands and many of the examples found in this manual. The two files are the employee master file, VC2EMP, and the department master file, VC2DPT. Both files are located in the VC2CLF library and are initially empty.

You should, prior to using any of the example programs, either

- create you own versions of these files into a library of your choosing. To create the physical files into the QTEMP library you can use these commands
  - CRTPF FILE(QTEMP/VC2EMP) SRCFILE(VC2CLF/QDDSSRC)
  - CRTPF FILE(QTEMP/VC2DPT) SRCFILE(VC2CLF/QDDSSRC)
  - depending on the example you want to run you may also need to create into your library one or more logical files. The instructions for creating these logical files are provided under the topics VC2EMP description and logical files and VC2DPT description and logical files.

or

- refresh both of these files in the VC2CLF library by first clearing them. To clear them you use the commands
  - CLRPFM FILE(VC2CLF/VC2EMP)
  - CLRPFM FILE(VC2CLF/VC2DPT)
- and then reload the two files as described under Loading Records into the Sample Files. Reloading the files will ensure that the results you see match what the documentation describes. Several of the CLF provided examples delete records, update fields, and perform other activities. By reloading the file prior to running the example programs you can reduce the potential for possible confusion.

## VC2EMP DESCRIPTION AND LOGICAL FILES

The DDS source for physical file VC2EMP can be found in member VC2EMP of the source file VC2CLF/QDDSSRC and is shown in Figure B. 1.

```
A.....T.Name+++++RLen++TDpB.....Functions+++++
                                         UNIQUE
R EMPRCD
  EMPNBR          5  0      TEXT('Employee Number')
* Status: F - Full time, P - Part time, I - Inactive
  EMPSTS          1      TEXT('Employee Status')
  EMPFNAME        40      TEXT('Employee First Name')
  EMPDPT          2      TEXT('Employee Department')
  EMPDPTM         2      TEXT('Department Managed')
  EMPEXT          4      TEXT('Employee Extension')
K EMPNBR
```

Figure B. 1 - DDS source used in creating the VC2EMP physical file

The file is created with a unique key based on the employee number field (EMPNBR). The record format is named EMPRCD and contains, in addition to EMPNBR, the fields employee status (EMPSTS), employee first name (EMPFNAME), employee department (EMPDPT), for managers the department the employee manages (EMPDPTM), and the employee telephone extension (EMPEXT).

The employee status can be one of three values – ‘F’ for full time, ‘P’ for part time, and ‘I’ for inactive. The department managed field contains blanks if the employee it not a manager.

For demonstration purposes there are five logical files built over the VC2EMP physical file. They are:

- VC2EMPDPT – VC2EMP keyed by employee first name within employee department
- VC2EMPEXT – VC2EMP keyed by employee telephone extension
- VC2EMPNAME – VC2EMP keyed by employee first name
- VC2EMPSTS – VC2EMP keyed by employee status

- VC2DPTEMPL – a multi-format logical file with one record format based on VC2EMP. The VC2EMP record format is keyed by employee number within employee department.

The source for each of these logical files can be found in a member by the same name in source file VC2CLF/QDDSSRC. To create these logical files into QTEMP you can use the following commands:

```
CRTL F FILE (QTEMP/VC2EMPDPT) SRCFILE (VC2CLF/QDDSSRC)
```

```
CRTL F FILE (QTEMP/VC2EMPEXT) SRCFILE (VC2CLF/QDDSSRC)
```

```
CRTL F FILE (QTEMP/VC2EMPNAME) SRCFILE (VC2CLF/QDDSSRC) +
      SRTSEQ (*LANGIDSHR)
```

```
CRTL F FILE (QTEMP/VC2EMPSTS) SRCFILE (VC2CLF/QDDSSRC)
```

```
CRTL F FILE (QTEMP/VC2DPTEMPL) SRCFILE (VC2CLF/QDDSSRC)
```

Note that when creating the VC2EMPNAME logical file SRTSEQ(\*LANGIDSHR) is to be specified. Database key fields, by default, are case sensitive. So to randomly read a record from the VC2EMPNAME file with a key value of 'Rebecca' you would have to request that record using exactly the same case. That is, requesting a record by key with a key argument of 'rebecca' or 'REBecca' will not compare equal to the database key value 'Rebecca'. Your program would get a record-not-found condition returned to it.

In order to simplify your use of some of the sample programs, SRTSEQ(\*LANGIDSHR) is used. Specifying SRTSEQ(\*LANGIDSHR) creates a logical file that, for key comparison purposes, will share the value of lowercase letters with the value of their uppercase equivalents. That is, 'rebecca', 'Rebecca', and 'REBECCA' will all be treated as if they were the same. The specification \*LANGIDSHR also indicates that you want this sharing done based on a language ID. The default language ID used is the LANGID job attribute associated with the job creating the logical file. To see what language ID your job is currently running with you can use the command

```
CHGJOB JOB (*)
```

Prompt the command with F4

Press F9 to see all keyword values

Page down until you find the entry for Language ID (typically the third panel)

To see what language IDs you could use, press F4 while the cursor is over your current language ID

The command

```
DSPJOB OPTION (*DFNA)
```

Will also show you the current language ID for your job, but not what other language IDs are available to you.

The VC2EMPNAME logical file as created when you installed CLF is based on the language ID ENU – US English.

When initially loaded by the RPG\_LOAD program the contents of VC2EMP, shown in arrival sequence order, are:

EmpNbr	EmpSts	EmpFName	EmpDpt	EmpDPTM	EmpExt
1	F	Rebecca		A1	9044
13	F	Bruce	A1	MN	4178
205	F	Amie	A1	CO	1759
107	F	Richard	MN	RO	9093
936	F	Hadleigh	MN		8266
4013	P	Matthew	MN		8254
4033	P	Dovid	MN		8266
4035	P	Moshe	MN		8266
521	P	Kiernan	CO		1759
656	I	Richard	A1		
40	F	Mindy	A1	MS	6896
52	I	Ruth	MN		5909
53	I	Wayne	A1		1234
1111	P	Clayton	RO		6736
1112	P	Betty	RO		6736

Table B. 1 - Contents of the VC2EMP database file

## VC2DPT DESCRIPTION AND LOGICAL FILES

The DDS source for physical file VC2DPT can be found in member VC2DPT of the source file VC2CLF/QDDSSRC and is shown in Figure B. 2.

```
A.....T.Name+++++RLen++TDpB.....Functions+++++
R DPTRCD
  DPTID          2          TEXT('Department ID')
  DPTNAME        30          TEXT('Department Name')
  DPTMGR          5  0      TEXT('Department Manager')
  DPTNXTMTG       Z          TEXT('Next Department Meeting')
  DPTMGRC         L          TEXT('Date of Last Mgr Change')
                        DATFMT(*USA)
  DPTSTRT         T          TEXT('Department Start Time')
                        TIMFMT(*HMS)
K DPTID
```

Figure B. 2 - DDS source used in creating the VC2DPT physical file

The record format is named DPTRCD and contains the fields department ID (DPTID), department name (DPTNAME), the employee ID for the manager of the department (DPTMGR), a timestamp recording the date and time of the next scheduled department meeting (DPTNXTMTG), a date field recording the date the department manager was last changed (DPTMGRC), and a time field recording the normal start time for employees of the department (DPTSTRT). The VC2DPT file is keyed by the department ID.

For demonstration purposes there is one logical file built over the VC2EMP physical file. The file is VC2DPTEMPL – a multi-format logical file with one record format that is based on VC2DPT. The VC2DPT record format is keyed by department ID. The source for VC2DPTEMPL can be found in a member by the same name in source file VC2CLF/QDDSSRC. You can create your own instance of the logical file into QTEMP using the command

```
CRTLF FILE(QTEMP/VC2DPTEMPL) SRCFILE(VC2CLF/QDDSSRC)
```

Though not used widely in the provided examples there are also two additional versions of the VC2DPT file. The first version, VC2DPTNUL, is used in demonstrating how to use null capable fields. VC2DPTNUL defines the fields DPTMGR, DPTNXTMTG, and DPTMGRC as being null capable. The second version of VC2DPT, VC2DPTVAR, is used in demonstrating how to use variable-length fields. VC2DPTVAR defines the field DPTNAME as being variable length. These files are not automatically created into the VC2CLF library. To create and load records into these two files refer to Instructions for

loading the VC2DPTNUL sample file and Instructions for loading the VC2DPTVAR sample file.

When initially loaded by the RPG\_LOAD program the contents of VC2DPT, shown in arrival sequence order, are:

DptID	DptName	DptMgr	DptNxtMtg	DptMgrC	DptStrT
A1	Corporate Management	1	2008-12-27- 10.00.00.000000	11/14/2008	08:00:00
MN	Product Development	13	2008-12-19- 15.00.00.000000	10/10/2008	07:00:00
CO	Product Support	205	2008-12-03- 11.30.00.000000	01/02/2008	06:30:00
MS	Administrative Support	40	2008-12-03- 11.00.00.000000	08/15/2008	08:00:00
RO	Product Marketing	107	2009-01-05- 08.00.00.000000	08/15/2008	07:30:00

*Table B. 2 - Contents of the VC2DPT database file*

## LOADING RECORDS INTO THE SAMPLE FILES

CLF provides the sample program RPG\_LOAD that you can use to load records into the provided VC2EMP and VC2DPT database files.

### ***Instructions for the RPG\_LOAD program***

You will find the source for the RPG\_LOAD program in member RPG\_LOAD of source file VC2CLF/VC2CLSRC. To compile the RPG\_LOAD program into QTEMP use the commands

```
ADDLIBLE LIB(VC2CLF)
CRTBNDCLF PGM(QTEMP/RPG_LOAD) SRCFILE(VC2CLSRC)
```

To load the records into the sample files ensure that you have an appropriate library list for the instances of the files you want to load and then use the command

```
CALL PGM(QTEMP/RPG_LOAD)
```

### ***Instructions for loading the VC2DPTNUL sample file***

The DDS source for the VC2DPTNUL database file can be found in member VC2DPTNUL of source file VC2CLF/QDDSSRC. To create the VC2DPTNUL sample database file into QTEMP use the command

```
CRTPF FILE(QTEMP/VC2DPTNUL) SRCFILE(VC2CLF/QDDSSRC)
```

The sample program DEV\_LOADNL loads the same five records into VC2DPTNUL that the RPG\_LOAD program loads into the VC2DPT database file. The only difference is that the department record for department MS will set the next department meeting field, DPTNXTMTG, to NULL.

You will find the source for the DEV\_LOADNL program in member DEV\_LOADNL of source file VC2CLF/VC2CLSRC. To compile the DEV\_LOADNL program into QTEMP ensure that you have an appropriate library list for locating your instance of the VC2DPTNUL file and then use the command

```
CRTBNDCLF PGM(QTEMP/DEV_LOADNL) SRCFILE(VC2CLF/VC2CLSRC)
```

To load the records into VC2DPTNUL use the command

```
CALL PGM(QTEMP/DEV_LOADNL)
```

## ***Instructions for loading the VC2DPTVAR sample file***

The DDS source for the VC2DPTVAR database file can be found in member VC2DPTVAR of source file VC2CLF/QDDSSRC. To create the VC2DPTVAR sample database file into QTEMP use the command

```
CRTPF FILE(QTEMP/VC2DPTVAR) SRCFILE(VC2CLF/QDDSSRC)
```

The sample program DEV\_LOADVL loads the same five records into VC2DPTVAR that the RPG\_LOAD program loads into the VC2DPT database file. The only difference is that the length of each department name field will be set the length of the blank trimmed department name.

You will find the source for the DEV\_LOADVL program in member DEV\_LOADVL of source file VC2CLF/VC2CLSRC. To compile the DEV\_LOADVL program into QTEMP ensure that you have an appropriate library list for locating your instance of the VC2DPTVAR file and then use the command

```
CRTBNDCLF PGM(QTEMP/DEV_LOADVL) SRCFILE(VC2CLF/VC2CLSRC)
```

To load the records into VC2DPTVAR use the command

```
CALL PGM(QTEMP/DEV_LOADVL)
```

# Appendix C. Compile Errors Using the Precompiler

When you compile using the CLF precompiler and ask for a compile listing the precompiler will generate a report. In this report will be informational and diagnostic messages generated by the precompiler. In addition severe diagnostic messages will be sent to your job log.

The generated report is created with:

- a file name which is the same as the CL source member being precompiled
- user data which is the name of the CLF precompiler command

When diagnostic messages with a severity of 30 or higher are found by the precompiler the IBM provided compiler command is not run. These errors must be corrected before the precompiler will run the appropriate IBM create command.

If the precompiler finds no errors greater than or equal to 30 then the IBM compiler is run. Note that the precompiler only verifies the correct use of CLF-related commands. Additional errors may be found by the IBM compiler.

If you asked for a compile listing a second report will be created. This report is generated by the IBM compiler. The IBM compiler may detect errors in your source program and report them with potentially confusing error messages. Below are some of these IBM error messages and what to look for in your source program.

CPD0304 – SUBR command not allowed.

CPD0305 – No matching SUBR command for ENDSUBR command.

Even though you may have no subroutines in your CL source program this error can be reported by the IBM provided compilers when you have a DO command in your CL source with no matching ENDDO command. Examine your use of DO commands and ensure that each has a matching ENDDO.

The CLF precompiler will often generate subroutines based on your use of CLF commands. The actual error, CPD0714 – No matching ENDDO command for DO command, can be found in the IBM compiler listing but may initially be missed due to several messages related to the use of subroutines.

# Appendix D. CLF Commands

This appendix provides links to the CLF command documentation located at <http://www.powercl.com/clf/clfcommands>. The commands are grouped by syntax style and listed in alphabetical order.

## RPG-LIKE SYNTAX STYLE

- [CHAIN - Random Retrieval from a File](#)
- [CLOSE - Close File](#)
- [DELETE - Delete Record](#)
- [EXFMT - Write/Then Read Record Format](#)
- [FILE - File Description Specification](#)
- [INCLUDE - Include CL Source using CLF](#)
- [INDS - Indicator Specification](#)
- [OPEN - Open File for Processing](#)
- [READ - Read Record](#)
- [READC - Read Next Changed Record](#)
- [READE - Read Record with Equal Key](#)
- [READP - Read Prior Record](#)
- [READPE - Read Prior Record with Equal Key](#)
- [SETGT - Set Greater Than](#)
- [SETLL - Set Lower Limit](#)
- [UNLOCK - Release Record](#)
- [UPDATE - Modify Existing Record](#)
- [WRITE - Create New Record](#)

## CL SYNTAX STYLE:

- [ACQDEVCLF - Acquire Device using CLF](#)
- [CHGNULACLF - Change Null Attribute using CLF](#)
- [CLOFCLF - Close File using CLF](#)
- [CRTBNDCLF - Create Bound Program using CLF](#)
- [CRTCLFMOD - Create Module using CLF](#)
- [CRTCLFPGM - Create Program using CLF](#)
- [DCLFCLF - Declare File using CLF](#)
- [DCLFKSCLF - Declare File Key Structure using CLF](#)
- [DCLINDCLF - Declare Indicators using CLF](#)
- [DLTRCDCLF - Delete Record using CLF](#)
- [FRCDTACLF - Force Data using CLF](#)
- [GENFFDCLF - Generate File Field Definitions using CLF](#)
- [GENFKSCLF - Generate File Key Structure Definitions using CLF](#)
- [GENINDCLF - Generate Indicator Definitions using CLF](#)
- [OPNFCLF - Open File using CLF](#)
- [POSDBFCLF - Position Database File using CLF](#)
- [READRCDCLF - Read Record using CLF](#)
- [RLSDEVCLF - Release Device using CLF](#)
- [RLSRCDCLEF - Release Record using CLF](#)
- [RTVFINFCLF - Retrieve File Information using CLF](#)
- [RTVNULACLF - Retrieve Null Attribute using CLF](#)

- [RTVVFLCLF - Retrieve Variable Field Length using CLF](#)
- [SETDEVCLF - Set Device using CLF](#)
- [SETVFLCLF - Set Variable Field Length using CLF](#)
- [UPDRCDCLF - Update Record using CLF](#)
- [WRKOPNFCLF - Work with Open Files using CLF](#)
- [WRTRCDCLF - Write Record using CLF](#)
- [WRTREADCLF - Write/Read Record using CLF](#)
- [WRTRRNCLF - Write Record by RRN using CLF](#)