Rocket | M204

Rocket Model 204 Application Development Guide

Programmer's Guide

Version 7 Release 4.0

May 2012
204-74-ADG-01

Rocket®

Notices

Edition

Publication date: May 2012

Book number: 204-74-ADG-01

Product version: Rocket Model 204 Application Development Guide

Copyright

Trademarks

License agreement

Contact information

Web Site: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451–1468
USA
Tel: +1.617.614.4321
Fax: +1.617.630.7100

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America	+1.800.755.4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

and click **Rocket M204.**

Contents

About this Guide

This guide describes naming conventions and recommends coding guidelines and locking management strategies to help you solve the problems inherent in a multiuser environment.

This guide is not designed to replace any manual in the Model 204 documentation set, but to provide a supplement that helps you produce clearer, faster applications. The information in this guide is not version-specific.

Audience

This guide is for system managers, file managers, and application programmers who want to develop Model 204 User Language code that is efficient, readable, and easy to maintain.

Model 204 documentation set

The complete commercially released documentation for the latest version of Model 204 is available for download from the Rocket M204 customer portal.

To access the Rocket Model 204 documentation:

1. Navigate to:

   http://www.rocketsoftware.com/m204

2. From the drop-down menu, select **Products > Model 204 > Documentation**.

3. Click the link to the current release and select the document you want from the list.

4. Click the .zip file containing the document.

5. Choose whether to open or save the document:
   - Select **Open** and double-click the pdf file to open the document.
   - Select **Save as** and select a location to save the zip file to.

Documentation conventions

This guide uses the following standard notation conventions in statement syntax and examples:

Convention	Description
TABLE	Uppercase represents a keyword that you must enter exactly as shown.
TABLE *tablename*	In text, italics are used for variables and for emphasis. In examples, italics denote a variable value that you must supply. In this example, you must supply a value for *tablename*.
READ [SCREEN]	Square brackets ([ ]) enclose an optional argument or portion of an argument. In this case, specify READ or READ SCREEN.
UNIQUE | PRIMARY KEY	A vertical bar (|) separates alternative options. In this example, specify either UNIQUE or PRIMARY KEY.
TRUST | NOTRUST	Underlining indicates the default. In this example, NOTRUST is the default.
IS {NOT | LIKE}	Braces ({ }) indicate that one of the enclosed alternatives is required. In this example, you must specify either IS NOT or IS LIKE.
item ...	An ellipsis (. . .) indicates that you can repeat the preceding item.
item ,...	An ellipsis preceded by a comma indicates that a comma is required to separate repeated items.
All other symbols	In syntax, all other symbols (such as parentheses) are literal syntactic elements and must appear as shown.
nested-key ::= *column_name*	A double colon followed by an equal sign indicates an equivalence. In this case, *nested-key* is equivalent to *column_name*.
Enter your account: sales11	In examples that include both system-supplied and user-entered text, or system prompts and user commands, boldface indicates what you enter. In this example, the system prompts for an account and the user enters **sales11**.
File > Save As	A right angle bracket (>) identifies the sequence of actions that you perform to select a command from a pull-down menu. In this example, select the Save As command from the File menu.
EDIT	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

1

Model 204 Naming Conventions

Establishing naming conventions at your site helps everyone involved in reading, writing, or updating Model 204 code. Consistent naming conventions makes interpretation of existing code easier for people other than the original developer.

In this chapter

This chapter has the following sections:

- Model 204 filenames
- Data set names
- Procedure names
- Fieldnames
- %Variables
- Statement labels
- Subroutines
- Other naming convention considerations

Model 204 filenames

Filenames are limited to 8 characters (7 for DOS systems) and must begin with a letter (A-Z).

The Model 204 restrictions are as follows:

- A filename cannot begin with any of the following:

CCA
SYS
OUT
TAPE

- A filename must contain only alphanumeric characters (A-Z, 0-9).
- Filenames can either describe the file content or conform to local file-naming standards. For example:

```
EMPLOYE
PER0001
```

Technical Support recommends that you adopt a file-naming standard for your site. This standard should reflect data about the file contents, such as the file number or file description, and link to specific entries in the DICTIONARY. For example:

Application ID:	PER
Numeric File ID:	0001
File Description:	Personnel Data
DDNAME/Model 204 Name:	PER0001

Model 204 files made up of multiple data sets or that are part of a group should follow a consistent naming scheme. For example:

```
CREATE FILE EMPLOYE FROM EMP000l. EMP0002...
CREATE PERM GROUP ACCT FROM ACTPROC.ACTDATA
```

Data set names

Follow these rules when naming the physical data set, as it is known to the operating system:

- Use an identifier to indicate that the file is a Model 204 file. Distinguish between live and dump files.
- Use a name that incorporates the Model 204 filename.
- Where applicable, use a name that contains an application or data identifier.
- Use a name that identifies whether the file is a test or a production file.
- To ease conversion from test to production systems, use the same Model 204 filename in all Model 204 copies at your site (for example, between test and production copies). This also simplifies file maintenance.
- If your site has both data and procedure files, clearly distinguish between the two.

For example, your site uses these identifiers:

Department	System within the Department
PER Personnel	BEN Benefits
ACT Accounting	GL General Ledger

You then set up your file system based on the identifiers:

M204 Filename	Type	Data Set Name
BEN0001	Live file (Test)	PER.TEST.BEN0001.M204
GL00011	Live file	ACT.PROD.GL00011.M204
BENDMP1	Dump file	PER.PROD.BEN0001.DUMP
BENPRC1	Proc file	PER.PROD.BENPRC1.M204
GLPRC3	Proc file	ACT.PROD.GLPRC3.M204

Procedure names

Procedure names can describe the business function that they support, such as ADDPART, CHGPART, and so on, or they can meet standards already in Place: XG2104, BA101A77, for example.

Document all procedure names in the DICTIONARY with appropriate entries.

If multiple subsystems share the same procedure file, use the subsystem name as part of the prefix to distinguish among procedures from different subsystems.

When using the application subsystem facility, you must use a prefix to distinguish between precompiled and nonprecompiled procedures. Technical Support recommends using P. and N. prefixes for precompiled and nonprecompiled procedures. For example:

P.DAILYAVG	N.TABLESRT
P.DEL.REC	N.OPEN.PROC
P.ADD.STORE	N.PER.SETUP

Fieldnames

Limit the length of fieldnames to 20 characters. This is long enough to allow a meaningful name, but not so long as to be unwieldy. An advantage to having a formal fieldname length limit is to simplify passing fieldnames in %variables where you must declare the length of passed arguments. If a fieldname never exceeds 20 characters, you can safely set the %variable length to 20 and not worry about truncating the fieldname.

The recommended characters for fieldnames are A-Z, 0-9, period (.), and underscore (_). Using any other special characters, especially spaces, makes the fieldname more difficult to read and isolate within a program.

When possible, call the same field by the same name if it exists in more than one file, especially if the files are likely to be grouped and used as a logical unit.

Never use reserved words in fieldnames. The reserved words in Model 204 are listed in Table 1-1.

Table 1-1. Model 204 Reserved Words

AFTER	END	OR	EQ
ALL	FROM	RECORD	GE
AND	IN	RECORDS	GT
AT	IS	TAB	LE
BEFORE	LIKE	THEN	LT
BY	NOR	TO	NE
COUNT	NOT	VALUE	
EACH	OCC	WHERE	
EDIT	OCCURRENCE	WITH	

%Variables

Keep %variable names under 20 characters and make them understandable to other users. For example:

```
%DATE = $DATE
%ACCT = Account number
%CLIENT.ID = Client ID number
```

When creating %variables to hold the values of global variables, use a G. prefix followed by the name of the global variable. For example:

```
%G.CTL = $GETG ('CTL')
```

Separate long or multipart names with periods. For example:

Data	**Fieldname**	**%Variable**
Client number	CLIENT.NO	%CLIENT.NO
Client address	CLIENT.ADDR	%CLIENT.ADDR

Statement labels

Limit statement labels to 20 characters and use them wisely. Use statement labels rather than line numbers in new applications.

However, Technical Support recommends that you do not try to modify a numbered application to include a mixture of statement numbers and statement labels. If you mix statement labels and statement numbers, or incorrectly modify an application, you might cause Model 204 to branch incorrectly.

To restrict the use of statement numbers at your site, set FOPT to X’80'. For more information, see the *Model 204 File Manager’s Guide*.

Statement labels must:

- Start with an alphabetic character (A-Z)
- Contain only A-Z, 0-9, period (.), or underscore (_)
- End with a colon and space (:)

For example:

```
FD.MARKETS:
    FIND ALL RECORDS FOR WHICH...

CT.COMP.REC:
    COUNT RECORDS IN FD.COMP.REC...

DATE_MATCH:
    FIND ALL RECORDS...
```

Use consistent names throughout a procedure. Technical Support recommends using standard prefixes for all applications such as FD. for FIND statements, CT. for COUNT statements, and so on. For example:

`FD.WELL.NAME:`	`FIND RECORDS`
`CT.WELL.NAME:`	`COUNT RECORDS`
`SRT.WELL.NAME:`	`SORT RECORDS`
`FDR.WELL.NAME:`	`FIND AND RESERVE RECORDS`

Statement labels are required under the following conditions:

- A statement that is referred to by another statement within a procedure, as in a COUNT RECORDS IN label statement.
- A statement that follows a FIND or STORE statement and does not have an END statement. Technical Support strongly recommends, however, using an END statement instead.

Although not required, statement labels may be useful in the following cases:

- FIND statements (except FIND ALL VALUES)
- FOR loops
- STORE

- ON
- Subroutines

Subroutines

Begin subroutine labels with the prefix SUB or with the first three characters of the subroutine name. Once your site picks a style, be consistent.

Simple subroutines

Begin the names of %variables unique to a subroutine with a %SUB prefix, or with the first three characters of the subroutine name. For example:

```
%SUB.WELL.DATE
%WEL.DATE
```

Complex subroutines

To avoid conflicts in subroutine names, prefix all subroutines that reside in a procedure file outside your current file with a name unique to the procedure file. For example, if you have a procedure file containing various subroutines used for printing out data and named PRTSUBS:

```
PRTSUBS.SALES
PRTSUBS.REGION
```

Distinguish between internal and external subroutines. Use, for example, an IN and an EX prefix to differentiate between the two:

```
IN.WELL.DATE
EX.PRT.RECS
```

Other naming convention considerations

Make screen, menu, and report names reflect the business function performed. For example, you may want to name a menu used to add employee information EMP.INFO.MENU or a quarterly sales report QTR.SALES.RPT.

When using a record type field to distinguish between different record types within the same file, abbreviate the rectype name, but keep it meaningful and descriptive of the record's content. This helps make the data self-descriptive.

For example, in the Model 204 demonstration database, there are two types of records in the CLIENTS file. They are RECTYPE = POLICYHOLDER, for the records containing information about the holder of the insurance policy, and RECTYPE = DRIVER for records about every driver listed on any insurance policy.

2

Recommended Coding Guidelines

Model 204 provides you with a versatile set of application development tools. This chapter provides suggestions on how to use Model 204 features to create efficient applications.

In this chapter

This chapter has the following sections:

Sample coding structure

Table 2-1 lists and describes elements of User Language code.

Table 2-1. Sample Coding Structure

Comments	General description of program function
Commands	If not precompiled; for example, UTABLE
BEGIN	
Declare %Variables	Declare all %variables
ON Units	Control errors, attention, locking conflicts; may be an INCLUDE statement
Screen definitions	Define instream and included screens
Initialize %Variables	Assign values, get globals
Dialog Control	Read screen and input commands, process PF keys, get input data, perform edits, make audit entries
Record Selection Control	Process Finds, Sorts, Lists
File Update Logic	Store, change, delete, add, commit
Exit Logic	Set globals, jump to start of current procedure, advance to next procedure, make audit entries
Subroutines	Define instream subroutines
Includes	Include non-instream subroutines and screen definitions
END	

Using comments

Use comments, but keep them concise. Comments must be preceded by at least one asterisk (*) .

Note: Do not use symbols that can be misconstrued as dummy strings (??, ?&, ?S), even within comments. If you do, Model 204 treats them as dummy strings and gives you unexpected results. Also, do not end comments with a hyphen, because this also comments out the line following the comment.

The following formats for comments are suggested:

```
*****************************************************
*   INCLUDE VALIDATION SUBROUTINE FOR PART NUMBER   *
*****************************************************
IN PROCLIB INCLUDE SUB.PART.VAL
```

Or,

```
/? INCLUDE VALIDATION SUBROUTINE FOR PART NUMBER ?/
IN PROCLIB INCLUD SUB.PART.VAL
```

Using commands

When using commands within procedures:

- Try to confine commands to the subsystem initialization procedure.
- Set UTABLE parameters in the login procedure; especially if most procedures require the same general server size. Change individual procedures as needed, but be sure to reset the parameters. Use the TIME REQUEST command to obtain suggested values for table settings to help optimize the performance.
- Keep track of any system-level parameters that you set; that is, RESET MCPU, MBSCAN, ERMX. You can monitor system-level parameters from the audit trail.
- If you change parameters before a program is executed, or before control is transferred to another subsystem, remember to change them back afterward (for example, UTABLE under the application subsystem facility). This helps avoid runtime execution errors.

Declaring %variables

Although Model 204 allows you to implicitly declare %variables, Technical Support strongly suggests that you declare all variables as one section of code at the beginning of a procedure or standard subroutine. To ensure that all %variables are declared, set VLEN to 0, or (for Version 2, Release 1.0 users) specify VARIABLES ARE UNDEFINED.

Use declaration statements to specify:

- Type of %variable
- %variable length and number of decimal places
- Number of elements (array %variables)
- Use of the NO FIELD SAVE feature (optional)

For calculations other than basic integer arithmetic, we recommend specifying numeric %variables as FLOAT.

When declaring array %variables, use index loops. Also, use the NO FIELD SAVE option whenever possible to save space and to indicate that the %variable will not be used as a fieldname. For example:

```
BEGIN
   VARIABLE %X IS FLOAT
   VARIABLE %NAME IS STRING LEN 20 ARRAY (10) NO FIELD
SAVE
   SCREEN SELECT
```

```
      PROMPT 'NAME' LEN 10 INPUT NAME LEN 20
   END SCREEN FOR %X FROM 1 TO 10
      READ SCREEN SELECT
      %NAME (%X) = %SELECT:NAME
   END FOR
END
```

Using $functions

The $functions that are supported by Model 204 are documented in the *Model 204 User Language Manual.* Many other $functions are available through the User Group and other sources. Test unsupported $functions thoroughly before you use them.

Call $functions like $DATE only once per request. All $functions incur some overhead, and because the date rarely changes during a run, it is more efficient to establish the value once and, if you need to reference the value again, place the result of the call in a %variable.

You can eliminate the need to use $SUBSTR to retrieve the first *n* characters of a value by assigning the original value to a variable of the correct length. This method makes the code more difficult to read and update, but it saves CPU time. For example:

```
BEGIN
%A IS STRING LEN 2
%B IS STRING LEN 10
%B = '1234567890
   %A = %B
   PRINT '%A = ' WITH %A
END
```

```
%A = 12
```

If a situation involves repeated calls to a User Language $function, you can save CPU time by resolving the $function to a %variable and then calling the %variable. For example, recode the following statement:

```
IF $SUBSTR (MAKE.1.3) EQ 'FOR' OR $SUBSTR(MAKE 1.3)
EQ 'COM'...
```

to read:

```
%MAKE = S$SUBSTR (MAKE.1.3)
IF %MAKE EQ 'FOR' OR %MAKE EQ 'COM'...
```

This saves time both by referencing the $function, $SUBSTR, and the field, MAKE, only once.

Using the IF statement

When using IF statements, always use the IF...THEN...END IF format. That is, always end the statement with an END IF, and always use the THEN. This makes the statement easier for other people to read and update.

Formatting IF statements

Format IF statements as follows:

```
IF condition THEN
.
.
.
END IF
```

Use consistent operator syntax, that is, do not mix symbols and abbreviations. For example:

```
IF AGE LT 25 AND WEATHER EQ 'RAINY' THEN
```

or

```
IF AGE < 25 AND WEATHER = 'RAINY' THEN
```

Indent the operational logic. For example:

```
IF %A EQ 'X' THEN
   PRINT %A
END If
```

Always use a continuation hyphen to segment compound conditionals. For example:

```
IF %A EQ 'X' AND -
   %B EQ 'Y' THEN
      %C = %A
END IF
```

Keep your IF statements as simple as possible:

- Avoid unnecessary statement label branching.
- Avoid using negative logic.

Evaluating IF statements

The condition following the IF statement is always evaluated to zero or nonzero. Therefore, it is more efficient to use the expression IF %N rather than IF %N NE 0 and, conversely, IF NOT %N rather than IF %N = 0.

If you are evaluating a number of not equal (or equal) comparisons against a known set of values, (that is, IF %A NE 'A' and %A NE 'B'...), then use the following syntax. This uses about half the CPU time as well as reducing the amount of needed NTBL, QTBL, and VTBL:

```
IF NOT $ONEOF (%A, 'A/B/C/D','/') THEN...
```

Using %variables in IF statements

IF statement expressions are rendered more readable by using meaningful %variable names, label names, or $functions such as $ONEOF or $INDEX to describe the condition. For example, the sample IF statement in the previous section is clearer when written as:

```
IF NOT $ONEOF (%VALID.DATA, 'A/B/C/D','/) THEN...
```

If you have a complex condition that is used several times in a procedure, you can create a fixed %variable and refer to the entire condition under that %variable name. For example:

```
CTRECS:
   COUNT RECORDS IN FDRECS
   IF NOT COUNT IN CTRECS THEN
      PRINT 'NO RECORDS FOUND'
   END IF
   .
   .
   .
   ********************************
   **   EVALUATE IF CONDITION     *
   ********************************
   %VALID.DATA = (COUNT IN CTRECS) AND (%FLD = 'VAL')
   IF %VALID.DATA THEN
   .
   .
   .
   IF %VALID.DATA AND %1 = 5 THEN
   .
   .
   .
```

Using the computed JUMP TO

If an IF statement threatens to get too complex, think about using a computed JUMP TO statement. Consider the next two examples.

Using ELSEIF without the JUMP TO:

```
%OPTION = OPT
   IF %OPTION EQ '1' THEN
```

```
      JUMP TO OPT1
   ELSEIF %OPTION EQ '2' THEN
      JUMP TO OPT2
   ELSEIF %OPTION EQ '3' THEN
      JUMP TO OPT3
   END IF
OPT1: ... process option 1
OPT2: ... process option 2
OPT3: ... process option 3
```

Using the computed JUMP TO:

```
%OPTION = OPT
   JUMP TO (OPT1, OPT2, OPT3) OPT
      **********************************
      ** INVALID OPTIONS FALL TO HERE **
      **********************************
      PRINT 'INVALID OPTION'
      JUMP TO CHECKED
      OPT1:  ... process option 1
      OPT2:  ... process option 2
      OPT3:  ... process option 3
      CHECKED: ... end of jump to
```

ELSE IF vs. ELSEIF

Model 204 uses both the ELSEIF and ELSE IF statements. The differences in use are as follows:

- ELSE IF functions as a nested IF statement. Note that for ELSE IF, each IF requires an END IF statement. For example:

```
IF A > B THEN
   A = A - 1
ELSE IF B > C THEN
   C = C + 1
END IF
END IF
```

- For ELSEIF, only one END IF is required. For example:

```
IF A > B THEN
   A = A - 1
ELSEIF A < B THEN
   A = A + 1
*****************
** ELSE A = B **
****************
END IF
```

Finding records efficiently

Use The Model 204 FIND statement to search for specified information in your database. In general, use the FIND on fields that are defined as KEY or:

```
NUMERIC RANGE
HASHED
SORT KEY (SFGES. SFLS)
ORDERED
```

If you must search on a nonkey field, try to pair the FIND with a Boolean AND using a key field as the other search criterion. The key field search is performed first, which means (ideally) that there are fewer records in the set requiring a Table B search. This helps minimize Table B access. You might also want to consider redefining fields so that they meet the more efficient search criteria.

In the following example, NAME is nonkey and RECTYPE is key. The RECTYPE = DRIVER records are evaluated first using the KEY indices; NAME is then ANDed by performing a Table B search on each record in the RECTYPE = DRIVER set.

```
FD NAME = JANSSEN AND RECTYPE = DRIVER
```

Think about other methods of shortening or eliminating Table B searches. Among them:

- Keep the found set small if you are doing a scan.
- When building a system, if you know that a field will be used to find sets of records, define the field as KEY or one of the types listed above.
- Like the nonkey search, pair IS PRESENT with a Boolean AND and a key field when doing a FIND to minimize the number of records that require a sequential Table B search
- Limit nonkey searches by setting the MBSCAN parameter.

Whenever possible, use the results of previous FIND statements. This eliminates repeating I/O and CPU processing time already done for the first FIND. The following statement shows how to access a previous FIND:

```
FIND ALL RECORDS IN label FOR WHICH fieldname = value
```

In situations where you perform FINDs using both %variables and constants, put the constant terms (that is, terms without %variables or %%variables) into a separate FIND that is only executed once. Let the %%variable FIND access the previously found set. For example:

```
BEGIN
FD.MAKE:
   FIND ALL RECORDS WHERE MAKE = FORD
   END FIND
****************************************
```

```
**  PROMPT FOR FIELDNAME =VALUE PAIR  **
****************************************
   %FIELD = $READ ('ENTER FIELDNAME:')
   %VALUE = $READ ('ENTER VALUE:')
FD.VALUE:
   FIND ALL RECORDS IN FD.MAKE FOR WHICH -
      %%FIELD = %VALUE
   END FIND
   FOR EACH RECORD IN FD.VALUE
      PRINT VIN
   END FOR
END
```

Field attributes that affect the way a file is accessed during a FIND are described in Table 2-2.

Table 2-2. Field Attributes Affected by Searches

Field Attribute	Description
Non-KEY, non-ORDERED	• Table A for field code validation (during compilation • Table B direct file search
KEY	• Table A for field code validation (during compilation) • Table C for information (hashed access) • If not single record entry Table D for list or bit map
ORDERED	• Table A for field code validation (during compilation) • Table D or B for tree access

The chart below can help you use FIND efficiently when you are writing requests or creating fields. The FIND is performed on the value '12345'.

The codes are as follows:

- B = Table B Search
- C = Table C Index
- R = Numeric Range Index
- O = Ordered Index

Consult the following table.

Table 2-3. FIND Statement Efficiencies

FIND syntax	Key	Key Ord Char	Key Ord Num	NKey Ord Char	NKey Ord Num	Key NR	NKey NR
=	C	C	C	O	O	C	B
IS (EQ)	B	B	O	B	O	R	R

Table 2-3. FIND Statement Efficiencies (Continued)

FIND syntax	Key	Key Ord Char	Key Ord Num	NKey Ord Char	NKey Ord Num	Key NR	NKey NR
IS GE	B	B	O	B	O	R	R
IS LE	B	B	O	B	O	R	R
IS GT	B	B	O	B	O	R	R
IS LT	B	B	O	B	O	R	R
IS BEFORE	B	O	B	O	B	B	B
IS AFTER	B	O	B	O	B	B	B
IS NUM	B	B	O	B	O	R	R
IS NUM GT	B	B	O	B	O	R	R
IS NUM BEFORE	B	B	O	B	O	R	R
IS NUM AFTER	B	B	O	B	O	R	R
IS IN RANGE	B	B	O	B	O	R	R
IS NUM IN RANGE	B	B	O	B	O	R	R
IS BETWEEN	B	B	O	B	O	R	R
IS NUM BETWEEN	B	B	O	B	O	R	R
(IS) LIKE	B	O	B	O	B	B	B
IS ALPHA	B	O	B	O	B	B	B
IS ALPHA GT	B	O	B	O	B	B	B
IS ALPHA BEFORE	B	O	B	O	B	B	B
IS ALPHA AFTER	B	O	B	O	B	B	B
IS ALPHA IN RANGE	B	O	B	O	B	B	B
IS ALPHA BETWEEN	B	O	B	O	B	B	B
IS PRESENT	B	B	B	B	B	B	B

Committing records

The COMMIT statement ends updating, commits the updated information to the file, and allows a checkpoint. Commit your records at the end of completed transactions.

The two forms of the COMMIT statement are:

COMMIT	Leaves lists and found sets of records intact. COMMIT can be used within a FOR EACH RECORD loop.
COMMIT RELEASE	Releases all found sets of records. You must use COMMIT RELEASE outside of a FOR EACH RECORD loop.

If your site updates records frequently and in high volume, be sure that your requests commit records at regular intervals. (Depending on the size of the records, commit no less than every 500 records as a rule of thumb.)

Releasing records

The RELEASE statement relinquishes control of a found set of records.

Use RELEASE as soon as you no longer need a found set, sorted set, or list.

The two forms of the RELEASE statement are:

RELEASE RECORDS IN *label*	Releases records in the found set at the specified statement label.
RELEASE ALL RECORDS	Releases records in all found sets, sorted sets, and lists.

It is generally safer to use the RELEASE RECORDS IN label syntax to avoid inadvertently releasing records that you are still processing.

Using the IN clause

The IN clause explicitly identifies the current file or group that is being processed. The advantage of using the IN clause is that you can retain the procedure file as the default file. In addition, using the IN clause helps clarify User Language code by explicitly naming the file being processed.

Because the IN clause allows you to specifically name a file or group, Technical Support recommends that you use this clause in:

- FIND statements
- INCLUDE statements
- LIST statements
- Commands

- STORE RECORD statements
- CLEAR LIST statements

Always use the IN clause when processing a multiple file application. For example:

```
OPEN DAILY
OPENC CLIENTS
   password
OPENC CLAIMS83
   password
BEGIN
FD.CLAIM:
   IN CLAIMS83 FIND ALL RECORDS WHERE...
```

Using lists

Technical Support recommends using lists to process inquiry and report procedures. Place the records on the list as soon as possible. If you are not updating records on the list, release the found set as soon as you have established the list using the RELEASE RECORDS statement.

Records placed on a list are not necessarily locked; and any lock associated with the records results from their original found set. Nor does placing records on a list unlock them (only a RELEASE RECORDS statement accomplishes this). Therefore, if you place records on a list and then release the found set, you can retain the list for future processing. In this case, however, you cannot guarantee that the records will remain unchanged between the FIND and any subsequent processing statements.

When processing multiple files, you must first clear the list using the IN filename clause to establish list context and to prevent invalid cross-reference errors. For example:

```
IN filename CLEAR LIST listname
```

Clear lists as soon as you are done processing them using one the following statements:

RELEASE ALL RECORDS	To clear all lists
COMMIT RELEASE	To clear all lists
CLEAR LIST listname	To clear a specified list
RELEASE RECORDS ON listname	To clear a specified list

Remove records from a list using:

```
REMOVE RECORDS IN label FROM LIST listname
```

```
REMOVE RECORDS ON listname 1 FROM listname2
```

Sorting records

When sorting records, keep the found set as small as possible. If you are sorting large sets of records, think about using a sorted file structure, ORDERED or FRV attribute, or even a sorting package.

When sorting numeric fields, you must specify the NUMERIC attribute. Try to keep numeric fields under 64 significant digits.

Technical Support recommends that you always specify whether you want to sort a field in ascending or descending order by specifying ASCENDING or DESCENDING in the SORT syntax for each fieldname. This helps make the code more explicit for future users.

You can sort records in found sets and on lists; however, records in a sorted set cannot then be placed on a list. Also, you cannot directly update sorted records; you must first perform a FIND or use the FOR RECORD NUMBER statement within a FOR loop.

If you do want to update sorted records, you can use the SORT k RECORD KEYS statement to generate a set of records sorted by specified keys. Because these records contains record numbers, you can update records in this set via the FOR RECORD NUMBER statement. For more information, see the *Model 204 User Language Manual*.

FOR processing begins with the current record in the sorted set.

Using subroutines

Use subroutines to separate frequently executed logic and to segment requests for better readability. Simple subroutines are the most efficient way to segment frequently executed code

Use comments within a subroutine to explain the purpose and function of the subroutine, inputs and output, and to name the place(s) from which it is called.

Declare the %variables specific to a subroutine before the subroutine statement to ease maintenance and readability.

To make the code easier to read and maintain, try not to nest subroutines; that is, do not call one subroutine from within another.

Similarly, avoid including procedures that include other procedures, because this makes locating all the source code virtually impossible. One exception to this is for sets of subroutines that are often included together; in this case, you can create a procedure that consists only of INCLUDE statements.

Table 2-4. Sample Subroutine Structure

Subroutine Element	Description
SUBROUTINE SUB.name (parameters)	Names a complex subroutine

Table 2-4. Sample Subroutine Structure

Subroutine Element	Description
label: subroutine	Names a simple subroutine
Comments	Describes functions including the name(s) of the calling program(s)
Declare %variables	Assigns %variables, gets globals
Process logic	Specifies any User Language statements that are used by multiple requests
Exit logic	Sets globals, controls %variables and results of computations, and so on
END SUBROUTINE	Ends the subroutine

To make complex subroutines easier to read, place the subroutine name and each %variable name on a separate line (except for the CALL statement). Remember to end each continued line with a hyphen. For example:

```
SUBROUTINE BINARY.SEARCH ( -
   %SRT.ARRAY    IS   STRING ARRAY (*)
   %VALUE        IS   STRING LEN 20
   %SUBSCRIPT    IS   FIXED OUTPUT)
```

Using %variables In complex subroutines

COMMON %variables, if properly used, can save NTBL and VTBL space. When naming COMMON %variables, use the subroutine name in which the %variable is first declared before each COMMON declaration. This helps to ensure that the %variable is not used inadvertently in other routines. For example:

```
DECLARE %PRINT.RECORDS.NRECS IS FIXED COMMON
```

When using string %variables in complex subroutines, follow these rules:

- For STRING SCALARS, always specify LEN rather than depending on the VLEN setting.
- For STRING ARRAYS, do not specify LEN; the length of the calling routine is always used.
- For arrays, if NO FIELD SAVE is specified on the formal parameter, then it must also be specified on the actual parameter.
- When using screen or database variable name %variables, you must convey the context to the subroutine through referencing (see below).

Note: Screen and Image items can be used as input into complex subroutines, but they cannot be used for output.

Using dummy strings in subroutines

For nonapplication subsystem facility subroutines that contain file or group-specific statements such as FOR EACH RECORD, FOR EACH VALUE, or FIND, but where the file or group can vary, use dummy strings on any line where the filename is needed. The following example shows mixed use of both ?? and ?& dummy strings to prompt for a filename and then use that file within the procedures. The use of dummy strings is governed by the SUB and PROMPT parameters. See the *Model 204 User Language Manual* and the *Model 204 Parameter and Command Reference* for more information about using dummy strings in procedures and for information about the SUB and PROMPT parameters.

This example consists of three procedures: SETNAME, which finds the filename; MAKELIST, which places the desired records on a list; and PRINT.REC, which prints the records.

The advantage of using dummy strings in this instance is that you can change both the file and the field value easily (with modifications, this example can also allow for changes in the fieldname).

```
***********************
** PROCEDURE SETNAME **
***********************
BEGIN
*********************************************************
**  SET GLOBAL TABLE WITH NAME OF FILE. IF FULL, QUIT  **
*********************************************************
SETGFT:
   IF SSETG ('FILE'.'??FILENAME') AND SSETG ('CON-
TINUE'.'Y') -
    THEN
      PRINT 'GLOBAL TABLE IS FULL'
       PRINT 'YOUR REQUEST HAS BEEN STOPPED'

  ENDIF
END MORE
************************************
**  IF IT PASSES, GO TO MAKELIST  **
************************************
IF CONTINUE = Y, MAKELIST

************************
** PROCEDURE MAKELIST **
************************
MORE
DECLARE LIST NAME.LIST IN FILE $&FILE
**********************
** PROMPT FOR NAME  **
**********************
FDNAME:
   IN $&FILE FD LASTNAME = ?$LASTNAME
END FIND
*****************************
**  PUT FOUND SET ON LIST  **
*****************************
   PLACE RECORDS IN FDNAME ON LIST NAME.LIST
****************************************
**  CALL SUBROUTINE TO PRINT RECORDS  **
****************************************
   CALL PRINT.RECORDS (LIST NAME.LIST)
   INCLUDE PRINT.REC

***************************
**  PROCEDURE PRINT.REC  **
***************************
SUBROUTINE PRINT.RECORDS (LIST NAME.LIST IN FILE ?&FILE)
FRPAI:
   FOR EACH RECORD ON LIST NAME.LIST
      PRINT ALL INFORMATION
   END FOR
END SUBROUTINE
```

3

Managing Record Locking

Record locking controls the access and maintains the integrity of the data within the record. A record-locking conflict occurs when multiple users try to access the same records in a file for update.

This chapter discusses record-locking conflicts and how to prevent and manage them.

In this chapter

This chapter has the following sections:

- Locking conflict example
- Types of record locks
- Using ON units
- Using FIND AND RESERVE
- Coding RETRY counters
- Ensuring file integrity
- Controlling record locking problems

Locking conflict example

The example below shows two requests that cause a record-locking conflict; user #2 cannot update the record until user #1 releases it.

Request #1:

```
BEGIN
%NAME = 'HADRIAN WALL'
FD.NAME:
   IN EMPLOYEES FIND ALL RECORDS FOR WHICH
      EMPLOYEE = %NAME
   END FIND
   FOR EACH RECORD IN FD.NAME
      PRINT JOB.TITLE AT 10 AND DIVISION AT 20
   END FOR
END
```

Request #2:

```
BEGIN
%NAME = 'HADRIAN WALL'
%JOB = 'HRD REP'
FD.NAME:
   IN EMPLOYEES FIND ALL RECORDS FOR WHICH
      EMPLOYEE = %NAME AND
      JOB.TITLE = %JOB
   END FIND
   FOR EACH RECORD IN FD.NAME
      CHANGE JOB.TITLE TO 'SR. HRD REP'
   END FOR
END
```

In the example shown above, user #2 receives the following message:

```
RECORD ENQUEUING CONFLICT
DO YOU WANT TO CONTINUE?
```

Types of record locks

Locking, or enqueuing, can occur in a variety of circumstances depending on the type of file activity:

- A share lock (SHR) occurs for users with read access. All users can read a record, but no user has update access to it.
- An exclusive lock (EXC) occurs for users who are updating records. When records are exclusively locked, no other user has any access.

User Language statements create record locks as described in Table 3-1.

Table 3-1. User Language locks

UL Statement	Non-TBO Files	TBO Files
FIND	SHR LOCK on found set	Same
FIND AND PRINT COUNT	SHR LOCK on found set	Same
FIND AND RESERVE	EXC LOCK on found set	Same
FIND WITHOUT LOCKS	No lock	Same

Table 3-1. User Language locks

UL Statement	Non-TBO Files	TBO Files
CHANGE fieldname ADD fieldname INSERT fieldname DELETE fieldname	EXC LOCK on current record in FOR loop until end of current loop	Additional EXC LOCK until COMMIT
DELETE RECORD	EXC LOCK on current record in FOR loop until end of current loop	Additional EXC LOCK until COMMIT
DELETE RECORDS IN	EXC LOCK on found set of records from the FIND	Same
STORE RECORD	EXC LOCK on single record	Additional EXC LOCK until COMMIT
PLACE RECORDS ON LIST	No lock	Same
SORT RECORDS	No lock	Same

Using ON units

ON units, which allow you to perform an action on certain conditions, can be used for a number of different purposes within Model 204 code.

Two ON unit statements can be used in situations where record-locking conflicts might occur:

ON FIND CONFLICT	Handles conflicts that occur during FIND or FOR EACH RECORD statements when used to retrieve records.
ON RECORD LOCKING CONFLICT	Handles all types of conflicts that might arise during an attempt to lock records.

To help manage record-locking conflicts, include either an ON RECORD LOCKING CONFLICT or an ON FIND CONFLICT unit in every request in which you issue a FIND. If a procedure contains both ON RECORD LOCKING CONFLICT and ON FIND CONFLICT units, the ON FIND CONFLICT statement takes precedence for FIND conflicts.

If a FIND statement fails, that is, triggers an ON unit, no locks are held for that statement. Therefore, if an ON FIND CONFLICT is triggered during a group find, record locks are released for all files in the group.

ON RECORD LOCKING CONFLICT units can be reset throughout a program to handle conflicts in different ways. Remember that the last executed ON unit remains active until either the next ON unit is executed or the end of the request is reached.

Use ON FIND CONFLICT or ON RECORD LOCKING CONFLICT to ensure that any attempt by a FIND statement to get share access triggers the ON unit, if the records desired are already held in exclusive status by some other user. The records are held in share status during FOR EACH RECORD loop processing unless they are updated, in which case the lock is changed to exclusive. If the records are to be updated, Technical Support recommends that you use an ON RECORD LOCKING CONFLICT unit, because ON FIND CONFLICT is not triggered when the share lock is changed to exclusive.

In the following example, Model 204 places the found set of records in share status but allows other users shared access as well. As a result, no one user can gain the exclusive access needed for updating. This method guarantees that data in these records cannot be changed while this procedure examines and displays them. When using techniques, be sure to release the records as soon as possible to allow access by other users.

```
BEGIN
   SET HEADER 1 'RECORD STATUS REPORT'
   SET HEADER 2
   NEW PAGE
.
.
.
ON FIND CONFLICT
   AUDIT 'LOCK CONFLICT WITH $RLCFILE $RLCUSR $RLCREC
   READ OFC
   **********************************
   **  WHERE OFC IS A SCREEN DEF   **
   ** INCLUDED IN PROCESSING ABOVE **
   **********************************
   IF %OFC:ANSWER EQ 'Y' THEN
      RETRY
   ELSE JUMP TO FINISH
   END IF
END ON
FD.ACCT:
   FIND ALL RECORDS
END FIND
CT.ACCT:
   COUNT RECORDS IN FD.ACCT
   IF COUNT IN CT.ACCT EQ 0 THEN
      JUMP TO FINISH
   END IF
   FOR EACH RECORD IN FD.ACCT
      PRINT ALL INFORMATION
   END FOR
RELEASE RECORDS IN FD.ACCT
SKIP 2 LINES
PRINT 'END OF REPORT'
FINISH:
   END
```

The next example still provides read-only access, but the records can be deleted or field values changed between the time the FIND statement is issued and the current time of processing. Note that the IF statement is used within the FOR EACH RECORD loop to validate that the record has not been deleted and that the original FIND criteria have not changed. If changes have occurred, then the body of the IF statement is not processed.

```
BEGIN
.
.
.
ON FIND CONFLICT
AUDIT 'RECORD LOCKING CONFLICT CANCELLED' WITH -
   $RLCFILE $RLCUSR $RLCREC
JUMP TO PRT.ERROR
END ON
FD.ACCT:
   FIND ALL RECORDS FOR WHICH ACCT = '23643'
END FIND
CT.ACCT:
   COUNT RECORDS IN FD.ACCT
   IF COUNT IN CT.ACCT EQ 0 THEN
      PRINT 'NO RECORDS FOUND'
      JUMP TO FINISH
   END IF
   ***************************
   **  PUT RECORDS ON LIST  **
   ***************************
   PLACE RECORDS IN FD.ACCT ON LIST ACCTS
   ********************
   **  RELEASE THEM  **
   ********************
   RELEASE RECORDS IN FD.ACCT
   FOR EACH RECORD ON LIST ACCTS
      IF TYPE EQ 'BILLABLE' THEN
         PRINT NAME
         %CALC = %CALC + AMOUNT + 2
         PRINT %CALC TO 20
      END IF
   END FOR
   JUMP TO FINISH
PRT.ERR:
   PRINT 'YOUR REQUEST IS CANCELED'
   WITH 'DUE TO A RECORD LOCKING CONFLICT'
FINISH:
   END
```

Using FIND AND RESERVE

FIND AND RESERVE locks the current set of records in exclusive mode and is, therefore, recommended when executing update requests.

However, Technical Support recommends that you observe the following prohibitions:

- Do not reserve records across terminal I/O (READ SCREEN, for example).
- Do not reserve records during single-threaded (batch) updating.

To update a large set of records without locking the entire set, place the records to be updated on a list, and then issue a FIND AND RESERVE on each record as you update it. Whenever you use the FIND AND RESERVE statement, be sure to release the records with one of the following statements:

- RELEASE
- COMMIT RELEASE

Coding RETRY counters

If a FIND AND RESERVE statement encounters a locking conflict, the ON unit can either retry or end the request. If you use RETRY, use it with a counter. Increment the counter each time you retry, and cancel the request after a certain number of times. This prevents an infinite retry loop.

If you use a RETRY counter more than once in a program, remember to reset the counter to zero before each FIND AND RESERVE statement. If the FIND is within an outer FIND loop, you must create a new counter to monitor these iterations as well.

The following example shows the code for a screen that appears when Model 204 encounters a record-locking conflict. This example sets a RETRY counter and cancels the RETRY after a specified number of attempts.

```
BEGIN
*******************************************************
** IN CASE OF RECORD CONFLICT ASK USERS IF THEY WANT **
**  TO TRY AGAIN. CANCEL AFTER 25 USER ATTEMPTS.     **
**  AUDIT EACH ATTEMPT TO THE JOURNAL.               **
**  ON RECORD LOCKING CONFLICT SCREEN                **
*******************************************************
SCREEN ORLC
TITLE 'UNABLE TO ACCESS REQUESTED RECORDS' AT 24 BRIGHT
SKIP 3 LINES
PROMPT 'CONFLICTING USER:' PROMPT USER
PROMPT '            FILE:' PROMPT FILE
PROMPT '          RECORD:' PROMPT RECORD
SKIP 2 LINES
PROMPT 'DO YOU WANT TO TRY AGAIN?' INPUT
```

```
   TRY ONEOF Y,N DEFAULT 'N'
END SCREEN
ORLC:
   ON RECORD LOCKING CONFLICT
   **************************
   ** WRITE LOG TO AUDIT TRAIL **
   ******************************
   %RECORD = $RLCREC
   %FILE = $RLCFILE
   %USER = $RLCUSR
   AUDIT = 'RECORD LOCKING CONFLICT DURING DEPT -
      PROCESSING' WITH %RECORD %FILE %USER
   *******************
   **  SET COUNTER  **
   *******************
   %COUNTER = %COUNTER + 1
   IF %COUNTER EQ 25 THEN
      BACKOUT
      PRINT '**** REQUEST CANCELLED ****'
      %COUNTER = 0
      JUMP TO FINISH
   END IF
   **************************************
   **  ASK IF USER WANTS TO TRY AGAIN  **
   **************************************
   %ORLC:RECORD = %RECORD
   %ORLC:FILE =   %FILE
   %ORLC:USER =   %USER
   READ SCREEN ORLC
   IF ORLC:TRY EQ 'Y' THEN
      RETRY
   ELSE
      PRINT '**** YOUR REQUEST HAS BEEN CANCELLED****'
      JUMP TO FINISH
   END IF
   END ON
   *****************************************
   **  FIND ACCT RECORDS FOR EACH DEPT  **
   ***************************************
   FD.ACCT:
      FIND AND RESERVE ALL RECORDS FOR WHICH -
         DEPT = ACCOUNT
      END FIND
   PROCESS:
      FOR EACH RECORD IN FD.ACCT
          CHANGE CORRECTED TO 'YES'
      END FOR
FINISH:
   END
```

Ensuring file integrity

If you are updating from records placed on a list, you can help ensure file integrity, and maintain the logical consistency of your data, by keeping the user ID of the person who performed the last update on record. Check to see that no other user has updated the record while you are updating.

In the following example, one record at a time is displayed for update. After the record is changed, the update ID is checked. If it is not the number expected, an error message is issued.

```
BEGIN
.
.
.
FD.VIN:
   IN VEHICLES FD VIN = 123456789
   END FIND
IN VEHICLES CLEAR LIST VIN
PLACE RECORDS IN FD.VIN ON LIST VIN
RELEASE RECORDS IN FD.VIN

UPD.SCR: FOR 1 RECORD IN FD.VIN
   %UPDATEID = UPDATE ID

**********************
**  FILL SCREEN HERE  **
************************
END FOR
IF $CHKTAG ('SCR.UPD.VIN') THEN
   REREAD SCREEN SCR.UPD.VIN
ELSE
   READ SCREEN SCR.UPD.VIN NO REREAD
END IF
.
.
.
************************
**  EDIT SCREEN HERE  **
************************
FDR.VIN:
   FDR VIN = 123456789
   END FIND
   FR FDR.VIN
      IF UPDATE ID = %UPDATEID THEN
         CHANGE UPDATE ID TO UPDATE ID + 1
         CHANGE ...(other record fields)
      .
      .
      .
      ELSE
         %SCR.EMSG = 'RECORD CONCURRENTLY UPDATED' -
         WITH 'REDO UPDATES ON REFRESHED RECORD'
         IN VEHICLES CLEAR LIST VIN
         PLACE RECORDS IN FDR.VIN ON LIST VIN
         RELEASE RECORDS IN FDR.VIN
         JUMP TO UPD.SCR
      END IF
   END FOR
COMMIT RELEASE
END
```

Controlling record locking problems

To control record-locking problems, keep in mind the following guidelines:

- You can avoid many conflicts by releasing records as soon as you no longer need them.
- Use a commit statement at the end of every update unit.
- If possible, keep update units within the same terminal I/O point.
- In a multi-request application, segregate updating functions from read-only functions.
- To free pages in CCATEMP, use the CLEAR LIST statement when you no longer need the records in a list.
- Perform REDEFINE, RENAME, and DELETE FIELD functions during off-peak hours.
- When not using the application subsystem facility, set the ENQRETRY parameter to control the number of retries before a message is generated.
- Place the records you need on a list, release the records, and process from the list in situations where no updates are taking place or where updates are known not to affect the data in question.
- Defer index updates whenever possible.

For more information, see the *Model 204 File Manager's Guide.*

Index