LAWSON

Lawson 4GL Application Program Interfaces

Version 9.0.1.x

Published May 2012

Document Number EAPI-901-U-03

Legal Notices

This content is for instructional or informational purposes only. This content may be changed or superseded without notice, and is not part of any Lawson product, maintenance or services warranty.

Export Notice: Pursuant to your agreement with Lawson, you are required (at your own expense) to comply with all laws, rules, regulations, and lawful orders of any governmental body that apply to you and the products, services or information provided to you by Lawson. This obligation includes, without limitation, compliance with the U.S. Foreign Corrupt Practices Act and all other applicable corrupt practices legislation (which prohibits certain payments to governmental officials and political parties), export control regulations, and regulations of international boycotts. Without limiting the foregoing, you may not use, distribute or export the products, services or information provided to you by Lawson except as permitted by your agreement with Lawson and any applicable laws, rules, regulations or orders. Non-compliance with any such law, rule, regulation or order shall constitute a material breach of your agreement with Lawson.

Intellectual Property: All brand or product names mentioned herein are trademarks or registered trademarks of Lawson, or the respective trademark owners. This documentation is the proprietary information of Lawson protected internationally under copyright and related intellectual property laws. Lawson customers or authorized Lawson business partners may use, copy, or transmit this document for their internal use only. Any other use or transmission requires advance written approval of Lawson.

Modifications to this Document: Lawson does not authorize changes to this document and does not warrant or provide maintenance for **any** modification, alteration or addition made to this document.

Contents

Chapter 1

Introduction

Overview

Lawson applications use common procedural code to

- reduce the programming effort during application development,
- reduce the application testing effort,
- create a standard interface for all application systems, and
- create a standard interface for intra- and inter-application communications.

This manual describes the procedural code defined and used globally in all Lawson application systems.

What are APIs?

The Application Program Interfaces (APIs) described in this document are Lawson 4GL routines that validate records, display information related to records, and create or update files in the database.

Most APIs only let you inquire on or validate data. The validation process returns a result flag related to the inquiry. The result flag indicates if the item is found, if it is valid, and so on. The inquiry process accesses and returns data such as name, address information, and default codes.

Components of an API

Each API has a library, a parameter definition, and an execute subroutine.

- A library contains the call to the API. For the APIs described in this manal, the library is usually defined in a procedure division library (pdlib) file.
- A parameter definition contains the definition of the parameters that the API uses. For most of the APIs in this manual, the parameters are defined in the working storage (wslib) source file.
- An execute subroutine is the procedure name of the API performed by the program that requests the API.

Each API has input and output parameters. The input and output parameter specification consists of a field name, a field type, and a description.

- The Field Name value is the parameter name that the called program recognizes.

- The Type field lists the parameter field type, followed by the parameter field size. The parameter field size includes the field length, the number of decimal positions, and a sign indicator.
- The Description field describes the field, has a value list, and indicates if the field is required.

The following sections provide details of the conventions used in the API descriptions.

- "Field Type Conventions" on page 11
- "Field Size Conventions" on page 11
- "Field Examples" on page 11

Field Type Conventions

The following table defines the types of fields allowed in an API.

A field type of	Means the field is
N	Numeric
A	Alphanumeric

Field Size Conventions

The following table defines the conventions used to specify a field size for an API.

A field size of	Means the field is
3	3 characters long.
7,5	7 characters long; 5 of the 7 characters are decimal positions.
S15,2	15 characters long, 2 of the 15 characters are decimal positions, and a sign is allowed, which means the field can be positive or negative.

Field Examples

The following table lists examples of field specifications for APIs.

A field value of	Means the field is
A 5	Alphanumeric; 5 characters long
N 4	Numeric; 4 characters long; no decimal positions
N 5,5	Numeric; 5 characters long; all are decimal positions
N S15,2	Numeric; 15 characters long; 2 of the 15 characters are decimal positions; the field can be positive or negative.

How This Manual Is Organized

This manual is a reference document for system and application programmers responsible for special user programs that require services from the global routines. It describes how to use each routine and the services provided by the routine.

This manual assumes that you are familiar with Lawson Software naming conventions, Lawson 4GL language, the COBOL programming language, and application definition techniques. For details, see *Application Development Workbench* and *Application Development Workbench Standards*.

Section	Contents
"Global Data Structures" on page 13	Describes each data structure and comments on any special programming considerations.
"Global Routines" on page 15	Describes each procedural library and each entry in that library. Routine inputs and outputs are described. Programming examples show how to use each routine.
"Date and Time Processing" on page 55	Provides routines for date and time processing.
"Batch Processing" on page 76	Provides routines that allow users to create and submit batch jobs using online and batch programs.
"Database Input/Output Routines" on page 91	Describes the generated input and output routines for database processing.
"Database Update and Deletion Routines" on page 122	Describes the routines used to update, modify, or delete database records.
"Database Index Filter Routines" on page 148	Describes the creation and use of database filters.
"CSV File Processing Routines" on page 173	Describes routines for processing comma-separated variable files, Examples show the use of each routine.
"Comment and URL Attachment Processing Routines" on page 184	Describes routines for attaching comments and URL and e-mail addresses to files.
"Vertex Quantum Routines" on page 197	Describes routines for determining the taxing jurisdiction and calculating sales and use taxes. Examples show the use of each routine.
"Deprecated APIs" on page 214	Describes APIs that you might see in the source code, but you should no longer use.

Chapter 2

Global Data Structures

This chapter describes the global data structures. Considerations in using the structure in a program, if any, are mentioned.

NOTE Each of the data structures described in this chapter is subject to change as the systems and applications grow. However, the data defined here will continue being defined. In future releases, some data currently defined in these library structures might be defined by the precompiler without the use of libraries.

BEGIN1, BEGIN2

The data structure in BEGIN1 defines the system date and time values in several formats. This library item must be the last item included in a COBOL shell DATA DIVISION because it contains the PROCEDURE DIVISION header.

BEGIN2 gets the current date and time and inserts them into the area defined by BEGIN1.

During the execution, the build shell (bldsh) utility automatically includes the contents of BEGIN1 and BEGIN2 in every generated program without the use of the library technique.

COMMON

The data fields in the COMMON library data structure define standard switches, standard values for error codes, parameters, and work and message areas available to all application source code.

This library is included in every application program. All routines, both global and local, expect to use fields defined here.

CRTWS

The two data structures defined in CRTWS describe the user interface and program control information needed by Lawson 4GL routines and the transaction input area. During terminal input, the data moves from this area to the form areas defined by the form generation process. For a discussion of the form generation process, see *Application Development Workbench Standards*.

During program execution, these data structures store the name of the active form and program form ID; security processing considerations; error messages, numbers, or both; variable text for error messages; job names; user names; and so on. This is the central control area for the user interface and transaction.

DATEWS

The data structure in **DATEWS** contains all work areas and result areas for date evaluations, comparisons, and expansions. Dates from application programs are moved here for processing and checking. This area is a working area as opposed to that described by **BEGIN1**. **BEGIN1** holds the date and time obtained when the program or transaction started, and is generally used as read-only.

This data structure is included in every application program that uses date evaluation routines. All date processing routines expect input into this area and leave the results in this area.

JOBWS

The data structure in JOBWS contains all work areas and result areas for batch job scheduling from within a application program.

This library is included in application COBOL shells that create, submit, or delete batch jobs.

STRWS

The data structure in STRWS is used by the collection of string routines in STRRTNS as input, work, and output areas. Although these routines are called by compiled source code to build derived elements, the routines and data areas are available to application programs for other uses.

All data defined in this structure should be considered volatile. This structure is included whenever a STRRTNS routine is referred to.

The report processing routines are the primary users of this area. If you use it for your own stringing requirements, you need to set up your parameters, call the string routines, and immediately move the results to your own area. Do not call any report functions between the start and finish of your source code.

Chapter 3

Global Routines

This chapter describes routines or APIs that are available to all application programs.

Error and Message Processing

Error and message processing services provide routines to display and print errors and messages.

780-DISPLAY-MSG

780-DISPLAY-MSG gets and displays an error message for errors associated with application program processing. Before entry, the caller sets the inputs in CRTWS. The error message (**CRT-MESSAGE**), as built, is left in CRTWS after displaying.

Library	BATCHERR
Input	**CRT-PROGRAM-CODE** **CRT-ERROR-CAT** If you do not explicitly set the error category, the API assumes that the error category is the same as the program code. **CRT-MSG-NBR** **CRT-ERROR-VARIABLES**
Output	**CRT-MESSAGE** (expanded message) **CRT-ERROR-CAT**(reset) **CRT-MSG-NBR** (reset) **CRT-ERROR-VARIABLES** (reset)

Programming Example

```
*       Set error message for program error display.
        MOVE error category             TO CRT-ERROR-CAT.
        MOVE error-number               TO CRT-MSG-NBR.
        MOVE "special text"             TO CRT-ERR-VAR1.
        PERFORM 780-DISPLAY-MSG.
```

780-PRINT-ERROR-MSG

The 780-PRINT-ERROR-MSG routine is used to retrieve and print an error message for errors associated with program parameter information. Before entry, the caller sets the inputs in CRTWS. The error message (**CRT-ERROR-MSG**), as built, is left in CRTWS after printing and displaying.

Library	BATCHERR
Input	**CRT-PROGRAM-CODE** **CRT-ERROR-CAT** If you do not explicitly set the error category, the API assumes that the error category is the same as the program code. **CRT-ERROR-NBR** **CRT-ERROR-VARIABLES**
Output	**CRT-ERROR-MSG** (expanded message) **CRT-ERROR-CAT** (reset) **CRT-ERROR-VARIABLES** (reset)

Programming Example

```
*       Set error message for parameter error.
        MOVE error-category          TO CRT-ERROR-CAT.
        MOVE error-number            TO CRT-ERROR-NBR.
        MOVE "special text"          TO CRT-ERR-VAR1.
        PERFORM 780-PRINT-ERROR-MSG.
```

780-PRINT-MSG

780-PRINT-MSG is gets and prints an error message for errors associated with an application program. Before entry, the caller sets the inputs in CRTWS. The error message (**CRT-MESSAGE**), as built, is left in CRTWS after printing and displaying.

Library	BATCHERR
Input	**CRT-PROGRAM-CODE** **CRT-ERROR-CAT** **CRT-MSG-NBR** **CRT-ERROR-VARIABLES**
Output	**CRT-MESSAGE** (expanded message) **CRT-ERROR-CAT** (reset) **CRT-MSG-NBR** (reset) **CRT-ERROR-VARIABLES** (reset)

Programming Example

```
*       Set error message for program error.
        MOVE error-category               TO CRT-ERROR-CAT.
        MOVE error-number               TO CRT-MSG-NBR.
        MOVE "special text"             TO CRT-ERR-VAR1.
        PERFORM 780-PRINT-MSG.
```

790-GET-ERROR-MSG

790-GET-ERROR-MSG routine retrieves a standard error message and builds the variable portion of it with program-supplied information (which may be null). Before returning to the caller, the variable area (**CRT-ERROR-VARIABLES**) and the application system code (**CRT-ERROR-CAT**) are re-initialized to spaces.

Library	DATABASE
Input	**CRT-PROGRAM-CODE**
	CRT-ERROR-CAT
	CRT-ERROR-NBR
	CRT-ERROR-VARIABLES
Output	**CRT-ERROR-MSG** (expanded message)
	CRT-ERROR-CAT (reset)
	CRT-ERROR-VARIABLES (reset)

Programming Example

```
*       Set error message for program error.
        MOVE system-code                  TO CRT-ERROR-CAT.
        MOVE error-number                 TO CRT-ERROR-NBR.
        MOVE "special text"               TO CRT-ERR-VAR1.
        PERFORM 790-GET-ERROR-MSG.
```

790-GET-MSG

790-GET-MSG retrieves a standard error message and to build the variable portion of it with program-supplied information (which may be null). Before returning to the caller, the error message number (**CRT-MSG-NBR**) is reset to 0 and the variable area (**CRT-ERROR-VARIABLES**) and the application system code (**CRT-ERROR-CAT**) are re-initialized to spaces.

Library	DATABASE
Input	**CRT-PROGRAM-CODE**
	CRT-ERROR-CAT
	CRT-MSG-NBR
	CRT-ERROR-VARIABLES
Output	**CRT-MESSAGE** (expanded message)
	CRT-ERROR-CAT (reset)
	CRT-MSG-NBR (reset)
	CRT-ERROR-VARIABLES (reset)

Programming Example

```
*       Set error message for program error.
*       Reset message number after the call.
        MOVE system-code                TO CRT-ERROR-CAT.
        MOVE error-number               TO CRT-MSG-NBR.
        MOVE "special text"             TO CRT-ERR-VAR1.
        PERFORM 790-GET-MSG.
```

File Processing

File processing APIs allow you to name, create, or remove temporary files and print files.

900-BUILD-PRINT-FILE-NAME

900-BUILD-PRINT-FILE-NAME builds a unique print file name and verifies that it does not already exist. The name (100 characters maximum) is returned in **CRT-PRINT-FILE**. This name must be created before the print file can be opened for output.

Library	DATABASE
Input	No input needed.
Output	**CRT-PRINT-FILE** (always set)

Programming Example

```
*       Build a print file name.
        PERFORM 900-BUILD-PRINT-FILE-NAME.
```

900-BUILD-TMP-FILE-NAME

900-BUILD-TMP-FILE-NAME builds a unique temporary file name and verifies that it does not already exist. The name (100 characters maximum) is returned in **WS-TMP-FILE**. This name must be created before the temporary file can be opened for output.

End users are allowed to change the default path of work files built by this routine. End users can change the path by using the Job Definition (**jobdef**) utility in either the header area (job level) or detail area (step level). Changing the path at the job level affects the entire job, while changing the path at the step level affects only that step.

Library	DATABASE
Input	No input required.
Output	**WS-TMP-FILE** (always set)

Programming Example

```
*       Build a temporary file name.
        PERFORM 900-BUILD-TMP-FILE-NAME.
```

900-FILEEXISTS

900-FILEEXISTS checks to see if a certain file exists.

Library	SYSCMDS
Input	**SYSCMD-FILENAME**
Output	**SYSCMD-ERROR**

Programming Example

```
IF (AR90F1-AR90A-NAME        NOT = SPACES)
    MOVE AR90F1-AR90A-NAME   TO SYSCMD-FILENAME
    PERFORM 900-FILEEXISTS
    IF (SYSCMD-ERROR         NOT = ZEROS)
        INITIALIZE AR90F1-AR90A-NAME
        MOVE "I"             TO AR90F1-FC.
```

900-GETLAWDIR

900-GETLAWDIR returns the value of the $LAWDIR environment variable .

Library	SYSCMDS
Input	No input needed.
Output	**SYSCMD-LAWDIR** **SYSCMD-ERROR**

Programming Example

```
        PERFORM 900-GETLAWDIR.

      *** NEXT LINE IS FOR USE WITH NT ***
       INSPECT SYSCMD-LAWDIR REPLACING ALL "\" BY "/".

       MOVE SYSCMD-LAWDIR          TO ICLAWDWS-LAWDIR.
```

900-GET-UNIQUE-ID

A unique ID is returned to the program in a string representation following industry standards used by DCE and Microsoft's implementation of GUID's. Each sub-field is converted to zero-filled hexadecimal digits with the values 10 through 15 represented as lowercase 'a' through 'f' respectively. Hyphens are inserted after

the current time in seconds, the lower 2 bytes of UserNbr, the 2 bytes of generation number, and the upper 2 bytes of an unsigned counter. The result is a 36-byte string field in the format of: 00000000-0000-0000-0000-000000000000.

Upon a successful call, the SYSCMD-ERROR will be zero and the **SYSCMD-UNIQUE-ID** will contain the unique ID. If the call fails, the **SYSCMD-ERROR** will be set to 1.

Library	SYSCMDS
Input	No input needed.
Output	**SYSCMD-UNIQUE-ID** **SYSCMD-ERROR**

Programming Example

```
*       Get Unique ID.
        PERFORM 900-GET-UNIQUE-ID.
```

900-MOVEFILES

900-MOVEFILES moves a file to a location that you choose.

Library	SYSCMDS
Input	**SYSCMD-FILENAME** **SYSCMD-TARGET**
Output	**SYSCMD-ERROR**

Programming Example

```
*       Move Files.
        MOVE FILENAMEFROM            TO SYSCMD-FILENAME.
        MOVE FILENAMETO             TO SYSCMD-TARGET.
        PERFORM 900-MOVEFILES
```

901-REMOVE-TMP-FILE

901-REMOVE-TMP-FILE deletes all the pieces of the temporary file whose name is passed as its argument.

Library	DATABASE
Input	**WS-TMP-FILE**
Output	No output returned.

Programming Example

```
*       Delete work file.
        MOVE temp-file-name             TO WS-TMP-FILE.
        PERFORM 901-REMOVE-TMP-FILE.
```

Locale Processing

Locale processing routines allow a program to add phrases to a language definition, find translations of phrases, and retrieve a locale definition for a report.

The translated phrase must fit within the character length defined, or the API returns the original English phrase, not the translation. The API does not truncate phrases.

900-ADD-PHRASE

900-ADD-PHRASE adds a phrase to a language definition if that phrase is not yet there. The phrase must be translated before it can be used by application programs.

Library	DATABASE
Output	No output returned.

Required Fields

Field	Type and length	Definition
CRT-PHRASE	A 50	English phrase used to retrieve the translated phrase.
CRT-PHRASE-SIZE	N 4	Size used to determine whether the short or long phrase will be retrieved. You cannot get back anything larger than the existing phrase, but you can truncate what you get back by passing a smaller value than the existing phrase. For more information on defining short and long phrases, see the *Lawson Administration: Translation* guide.

Programming Example

This routine adds a phrase to the system language definition.

```
MOVE phrase-to-add                  TO CRT-PHRASE.
MOVE phrase-length                  TO CRT-PHRASE-SIZE.
PERFORM 900-ADD-PHRASE.
```

900-GET-LOCALE-PHRASE-XLT

900-GET-LOCALE-PHRASE-XLT allows a developer to retrieve a translated phrase by using a specified locale. To retrieve phrases by using the locale set in the user's profile, use 900-GET-COL-PHRASE-XLT.

Library DATABASE

Required Fields

Field	Type and length	Definition
CRT-PHRASE	A 50	English phrase used to retrieve the translated phrase.
CRT-PHRASE-SIZE	N 4	Size used to determine whether the short or long phrase will be retrieved. You cannot get back anything larger than the existing phrase, but you can truncate what you get back by passing a smaller value than the existing phrase. For more information on defining short and long phrases, see the *Lawson Administration: Translation* guide.
CRT-PUT-DOTS	A 1	Y = Adds leader dots to any trailing space after the translated phrase. N = Does not add leader dots. When changed to N, it remains at N until the value is reset to Y.
CRT-LOCALE	A 10	Locale used to retrieve the translated phrase.

Return Value

Field	Type and length	Definition
CRT-PHRASE-XLT	A 50	The translated phrase. If the routine cannot find the locale, the locale's language, or the translated phrase, then the English phrase is passed back in the **CRT-PHRASE-XLT** field instead of a translated phrase. The API never returns an error; it either retrieves the translation, or passes back the English phrase.

Programming Example

This example translates a phrase to target language.

```
MOVE DMLODWS-LOCALE-RECORD (I7)
                           TO CRT-LOCALE
MOVE DMLODWS-ALIAS-NAME    TO CRT-PHRASE
MOVE 50                    TO CRT-PHRASE-SIZE
MOVE "N"                   TO CRT-PUT-DOTS
PERFORM 900-GET-LOCALE-PHRASE-XLT
MOVE DMLODWS-MEMBER-HANDLE
                            TO WSESB-MEMBER-HANDLE
MOVE DMLODWS-ALIAS-RECORD (I7)
                             TO WSESB-ALIAS-TABLE-NAME
MOVE CRT-PHRASE-XLT    TO WSESB-ALIAS-NAME
```

900-GET-LOCALE-COL-PHRASE-XLT

900-GET-LOCALE-COL-PHRASE-XLT allows a developer using a specified locale to retrieve a translated phrase in columns. Use this API when the translation has columns whose width requires the text to be split into multiple rows. To retrieve phrases in columns when using the locale set in the user's profile, use "900-GET-COL-PHRASE-XLT" on page 29

Library DATABASE

Required Fields

Field	Type and length	Definition
WS-COL-PHRASE-ROW-COUNT	N 4	Indicates the maximum number of rows in the translation.
WS-COL-PHRASE-COL-WIDTH	N 4	Maximum number of characters in a row.
CRT-COL-PHRASE-ROW	A 50	One row of the English phrase used to retrieve the translated phrase. Occurs up to 5 times. Number of characters in all occurrences can not exceed 50.
CRT-COL-PHRASE-XTL-ROWS	A 50	One row of the translated phrase. Occurs up to 5 times. Number of characters in all occurrences can not exceed 50. If the routine cannot find the locale, the locale's language, or the translated phrase, or it finds incorrect values for rows or width, the English phrase is passed back in the **CRT-COL-PHRASE-XLT** field instead of a translated phrase. The API never returns an error; it either retrieves the translation, or passes back the English phrase.
CRT-LOCALE	A 10	Locale used to retrieve the translated phrase.

Programming Example

This example retrieves a phrase.

```
MOVE 3                      TO WS-COL-PHRASE-ROW-COUNT.
MOVE 8                      TO WS-COL-PHRASE-COL-WIDTH.
MOVE first-row              TO CRT-COL-PHRASE-ROW(1).
MOVE second-row             TO CRT-COL-PHRASE-ROW(2).
MOVE third-row              TO CRT-COL-PHRASE-ROW(3).
MOVE locale-name            TO CRT-LOCALE.
PERFORM 900-GET-LOCALE-COL-PHRASE-XLT.
```

900-GET-NEXT-LOCALE

The 900-GET-NEXT-LOCALE routine retrieves the next locale from the LOCALE database file.

Library DATABASE

Required Field and Return Value

Field	Type and length	Definition
CRT-LOCALE	A 30	Locale is used to retrieve the next locale. The locale is also returned in this field.

Programming Example

This example gets the next locale.

```
MOVE locale     TO CRT-LOCALE.
PERFORM 900-GET-NEXT-LOCALE.
```

900-GET-PREV-LOCALE

900-GET-PREV-LOCALE retrieves the previous locale from the LOCALE database file.

Library DATABASE

Required Field and Return Value

Field	Type and length	Definition
CRT-LOCALE	A 30	Locale is used to retrieve the previous locale. The locale is also returned in this field.

Programming Example

This example gets the previous locale.

```
MOVE locale       TO CRT-LOCALE.
PERFORM 900-GET-PREVIOUS-LOCALE.
```

900-GET-PHRASE-XLT

900-GET-PHRASE-XLT finds the translation for a specified phrase and returns it to the caller. If the user has only one base language defined, the routine returns the input phrase as the result.

The routine retrieves a phrase by using the locale set in the user's profile. To retrieve phrases by using a specified locale, use the routine "900-GET-LOCALE-PHRASE-XLT" on page 25

Library DATABASE

Required Fields

Field	Type and length	Definition
CRT-PHRASE	A 50	English phrase used to retrieve the translated phrase.
CRT-PHRASE-SIZE	N 4	Size used to determine which phrase will be retrieved if it is available. If the user passes in a size greater than the PHRASE define (50 characters), then the PHRASE define will be used in its place. For more information on defining short and long phrases, see the *Lawson Administration: Translation* guide
CRT-PHRASE-XLT	A 50	Phrase translation buffer that will be populated with the translated phrase by the routine.
CRT-PUT-DOTS	A 1	Y = Adds leader dots to any trailing space after the translated phrase. N = Does not add leader dots. The default is Y.

Returned Value

Field	Type and length	Definition
CRT-PHRASE-XLT	A 50	The translated phrase. If the routine cannot find the default locale, the locale's language, or the translated phrase, then the English phrase is passed back in the **CRT-PHRASE-XLT** field instead of a translated phrase. The API never returns an error; it either retrieves the translation, or passes back the English phrase.

Programming Example

This example returns a phrase translation (from HR105).

```
PERFORM 840-FIND-PADSET1.
MOVE PAD-ITEM-NAME          TO CRT-PHRASE.
MOVE WS-PHRASE-SIZE         TO CRT-PHRASE-SIZE.
MOVE "N"                    TO CRT-PUT-DOTS.
PERFORM 900-GET-PHRASE-XLT.
MOVE CRT-PHRASE-XLT         TO SR-ITEM-NAME.
```

900-GET-COL-PHRASE-XLT

900-GET-COL-PHRASE-XLT finds the translation for a specified phrase and returns it to the caller in columns. If the user has only one base language defined, the routine returns the input phrase as the result.

The 900-GET-COL-PHRASE-XLT routine retrieves a phrase by using the locale set in the user's profile. To retrieve phrases in columns using a specified locale, use the routine "900-GET-LOCALE-COL-PHRASE-XLT" on page 26

Library DATABASE

Required Fields

Field	Type and length	Definition
WS-COL-PHRASE-ROW-COUNT	N 4	Maximum number of rows in the translation.
WS-COL-PHRASE-COL-WIDTH	N 4	Maximum number of characters in a row.
CRT-COL-PHRASE-ROW	A 25	One row of the English phrase used to retrieve the translated phrase. Occurs up to five times. Number of characters in all occurrences cannot exceed 50.

Field	Type and length	Definition
CRT-COL-PHRASE-XTL-ROWS	A 25	One row of the translated phrase. Occurs up to five times. Number of characters in all occurrences cannot exceed 50. If the routine cannot find the locale, the locale's language, or the translated phrase, or it finds incorrect values for rows or width, the English phrase is passed back in the **CRT-COL-PHRASE-XLT** field instead of a translated phrase. The API never returns an error; it either retrieves the translation, or passes back the English phrase.

Programming Example

This example translates a column phrase to the target language.

```
MOVE 3                      TO WS-COL-PHRASE-ROW-COUNT.
MOVE 8                      TO WS-COL-PHRASE-COL-WIDTH.
MOVE first-row              TO CRT-COL-PHRASE-ROW(1).
MOVE second-row             TO CRT-COL-PHRASE-ROW(2).
MOVE third-row              TO CRT-COL-PHRASE-ROW(3).
PERFORM 900-GET-COL-PHRASE-XLT.
```

900-GET-REPORT-LOCALE

900-GET-REPORT-LOCALE allows a developer to retrieve a locale definition for a batch report. Retrieving a locale definition ensures that the information in a batch report has the expected formatting and punctuation for the locale in effect at a particular time. Examples include formatting for date, time, and decimal numbers.

Use this routine when a batch report has rule-based translation enabled. Rule-based translation is used to change locales while the report is running based on the locale preference of a vendor, customer, and so on.

NOTE The 900-GET-REPORT-LOCALE routine can only be used in batch report programs

The 900-GET-REPORT-LOCALE routine requires no fields as input. Calling the routine adds the working storage for the report locale to the program. As a result, the program contains the working storage for both the report locale and the user locale.

Library	LOCALERTNS
Input	No input required.

Output Fields

Field	Type and length	Definition
RPT-LANGUAGE	A 10	The language or dialect specified in the locale definition.
RPT-DATE-SEPARATOR	A 1	The character used to separate day, month, and year in a date value. The default is the forward slash (/).
RPT-TIME-SEPARATOR	A 1	The character used to separate seconds, minutes, and hours in a time value. The default is the colon (:).
RPT-DATE-DISPLAY-FORMAT	A 6	Indicates the date format: MMDDYY DDMMYY YYMMDD The default is MMDDYY.
RPT-TIME-FORMAT	A 1	Indicates the time format: 1 = 12 hour 2 = 24 hour The default is 1.
RPT-THOUSANDS-SEPARATOR	A 1	The character used to delimit thousands in a number. The default is the comma (,).
RPT-DECIMAL-SEPARATOR	A 1	The character used to separate whole numbers from fractions in currency values. The default is the period (.).
RPT-PERCENT-SIGN	A 1	The character used to indicate percentages. The default is the percent sign (%).
RPT-CURRENCY-SYMBOL	A 1	The character used for currency values. The default is the dollar sign ($).
RPT-CURRENCY-SYMBOL-LOC	A 1	Indicates the placement of the currency symbol. P = Prefix (currency symbol to the left of the value) S = Suffix (currency symbol to the right of the value) The default is P.
DECIMAL-SIGN-LOC	A 1	Indicates the placement of the negative sign (-). P = Prefix (negative sign to the left of the value) S = Suffix (negative sign to the right of the value) The default is P.

Programming Example

```
PERFORM 900-GET-REPORT-LOCALE.
IF (HRFNF1-DATE-8-FIELD NOT = ZEROS)
    MOVE HRFNF1-DATE-8-FIELD TO HRFNWS-DATE-IN
    IF (RPT-DATE-DISPLAY-FORMAT = "MMDDYY")
        STRING HRFNWS-MM HRFNWS-DD HRFNWS-CC HRFNWS-YY
                            DELIMITED BY SIZE
                            INTO HRFNF1-VALUE
    END-IF
```

900-IS-VALID-LOCALE

900-IS-VALID-LOCALE validates a LOCALE. 900-IS-VALID-LOCALE checks the LOCALE table for valid records.

LOCALE names are uppercase only. Input given in mixed case is converted to all uppercase when checking against the LOCALE table records.

Library DATABASE

Required Field

Field	Type and length	Definition
CRT-LOCALE	A 30	Locale to be verified.

Return Value

Field	Type and length	Definition
CRT-IS-VALID-LOCALE	A 1	Y = The locale is valid. N = The locale is invalid. **NOTE** SPACES is a valid locale and returns Y. It represents the default locale set up at a given location.

Programming Example

Before using this API, make sure that any needed language codes have been created through the Language Definition utility (langdef).

NOTE Spaces indicate the base language at the client site.

```
IF (AR19F4-ANT-LANGUAGE-CODE (I1) NOT = SPACES)
MOVE AR19F4-ANT-LANGUAGE-CODE (I1) TO CRT-LOCALE
PERFORM 900-IS-VALID-LOCALE
IF (CRT-IS-VALID-LOCALE NOT = "Y")
    MOVE 110                        TO CRT-ERROR-NBR
    MOVE AR19F4-ANT-LANGUAGE-CODE-FN (I1)
                                    TO CRT-FIELD-NBR
```

Print Processing

Print processing services allow the program to control how reports print data. Before using these routines, you should be familiar with the information in the section "Print Routine Considerations" on page 34.

Print Routine Considerations

When using a print routine, you should consider the following:

- Before performing a print routine, the report program must move a print request value (a report group name) to **RPT-GROUP-REQUEST**.
- The page number is moved to the print line if “page” occurs in columns 123-126.
- The system date is moved to the print line if “date” occurs in column 8 and rpt-skip = 9.
- The system time is moved to the print line if “time” occurs in columns 8-11.
- Dates (in the format 99/99/99) are assumed to be in yymmdd sequence, and are displayed as mmddyy, yymmdd, or ddmmyy depending on the value in the **WS-DATE-DISPLAY-FORMAT** field.

STEPS **To print report data**

1. Move data to the fields in the group.

   NOTE You cannot move a value to the following types of fields because the system supplies their values. Even though you may see DATE, TIME, and PAGE fields in a report definition, you cannot access these fields.

   ```
   TD     Today's Date
   CT     Current Time
   PN     Page Number
   ```

   Move data to all other fields every time you print a group or the fields might contain erroneous data.

2. Move the group name to **RPT-GROUP-REQUEST**.
3. Perform 700-PRINT-RPT-GRP.

700-PRINT-RPT-GRP

700-PRINT-RPT-GRP builds a report page, then prints each line. Use this routine when you want the system to control the details of report appearance (such as line format, line advancing, page breaks, and new page headers) rather than controlling those details yourself.

Report distribution services allow reports to be sent to users under control of the program.

Library	This routine is not in a library. It is part of a template and is included in your source program when you compile.

Input	Report group number
Output	Printed lines

Programming Example

This example prints a report group.

```
IF (WS-EMP-GOOD)
    WRITE EVS-SSN-REC       FROM EVS-EVS-SSN-REC
    ADD 1                   TO WS-COUNT
    MOVE GN2D-EMP-EMPLOYEE  TO RPT-GROUP-REQUEST
    PERFORM 700-PRINT-RPT-GRP.
```

900-DISTRIBUTE-REPORT

900-DISTRIBUTE-REPORT copies a report to a specified user directory, prints it, or both, based on report parameter data describing a distribution group and the report.

Library	DATABASE
Input	**CRT-JOB-USER** **CRT-JOB-NAME** **CRT-STEP-KEY** **WS-DB-PRODUCT-LINE** **CRT-TOKEN** **CRT-RDST-WORK-FILE** **CRT-RDST-REPORT-NAME** **CRT-RDST-PRINTER** **CRT-RDST-DIST-GROUP** **CRT-RDST-NBR-COPIES** **CRT-RDST-SAVE-REPORT**
Output	The report stores or prints one or more times based on the specifications in the group list.

Programming Example

```
MOVE REPORT1-WF-NAME        TO CRT-RDST-WORK-FILE.
MOVE REPORT1-NAME           TO CRT-RDST-REPORT-NAME.
MOVE PWT-DIST-GROUP         TO CRT-RDST-DIST-GROUP.
MOVE PWT-PRINTER            TO CRT-RDST-PRINTER.
MOVE PWT-NBR-COPIES         TO CRT-RDST-NBR-COPIES.
MOVE PWT-SAVE-REPORT        TO CRT-RDST-SAVE-REPORT.
PERFORM 900-DISTRIBUTE-REPORT.
```

User Name Processing

The following user name routines validate, provide use and display options, and provide for long user name identification.

900-EDIT-USER-NAME

900-EDIT-USER-NAME edits the name in **WS-USER-NAME**and validates that it is in the list of names in the security tables. For an invalid name, **CRT-ERROR-NBR** is set to 021.

Library	DATABASE
Input	**WS-USER-NAME**
Output	**CRT-ERROR-NBR** (set for error)

Programming Example

```
*       Validate current user name.
*
        MOVE ZERO                       TO CRT-ERROR-NBR.
        MOVE new-user-name              TO WS-USER-NAME.
        PERFORM 900-EDIT-USER-NAME.
```

900-GET-LONG-USER-NAME

900-GET-LONG-USER-NAME gets the long user name, if it exists, from the security system. This is used most often to display the full first and last name of the user. The long name is found through the password of the passed **WS-USER-NAME**. If the security system is not used, or there is no long name available, **WS-LONG-USER-NAME** is returned with blank fill.

Library	DATABASE
Input	**WS-USER-NAME**
Output	**WS-LONG-USER-NAME** (set or reset)

Programming Example

```
        IF  (BL15F1-FC                  = "A")
        OR  (BL15F1-BOP-NAME            = SPACES)
             MOVE BL15F1-BOP-OPERATOR    TO WS-USER-NAME
             PERFORM 900-GET-LONG-USER-NAME
             MOVE WS-LONG-USER-NAME      TO BL15F1-BOP-NAME.
```

900-GET-USER-DISPLAY-NAME

900-GET-USER-DISPLAY-NAME displays a 30-character user identification name.

Library DATABASE

Input

Field	Type and length	Definition
WS-USER-DBUIDKEY	A 10	Finds the user ID name.

Output

Field	Type and length	Definition
WS-USER-DISPLAY-NAME	A 30	Displays the longer user name.

Programming Example

```
IF (APUSER-NOTFOUND)
OR (APJ-USER-CLASS NOT = AP04F2-APJ-USER-CLASS)
    INITIALIZE AP04F2-PT-APJ-USER-ID
ELSE
          MOVE APJ-USER-ID       TO WS-USER-DBUIDKEY
          PERFORM 900-GET-USER-DISPLAY-NAME
          MOVE WS-USER-DISPLAY-NAME
                         TO AP04F2-PT-APJ-USER-ID
END-IF
```

900-GET-USER-DBUIDKEY

900-GET-USER-DBUIDKEY retrieves the user identification name and converts it to or from a 30-character display.

Library DATABASE

Input

Field	Type and length	Definition
WS-USER-DISPLAY-NAME	A 30	Retrieves the long user name.

Output

Field	Type and length	Definition
WS-USER-DBUIDKEY	A 10	Converts to 10 characters.

Programming Example

```
MOVE AP04F2-APJ-USER-CLASS         TO APJ-USER-CLASS.
MOVE AP04F2-APJ-USER-ID (I1)        TO WS-USER-DISPLAY-NAME.
      PERFORM 900-GET-USER-DBUIDKEY.
MOVE WS-USER-DBUIDKEY               TO APJ-USER-ID.
MOVE AP04F2-APJ-INQUIRE-ACCESS (I1) TO APJ-INQUIRE-ACCESS.
MOVE AP04F2-APJ-UPDATE-ACCESS (I1)  TO APJ-UPDATE-ACCESS.
```

Security Processing

Security services allow company, accounting unit, and processing level security to be verified by the program.

900-ACCT-UNIT-SECURED

900-ACCT-UNIT-SECURED verifies that the current user has security access to this accounting unit. If the current user is secured from this accounting unit in the specified company, **CRT-ACCT-UNIT-SECURED** is set to Y; otherwise, it is set to N.

For a discussion of security processing, see *Lawson Administration: User Setup and Security*.

Library	DATABASE
Input	**CRT-COMPANY**
	CRT-ACCT-UNIT
Output	**CRT-ACCT-UNIT-SECURED** (always set)

Programming Example

```
MOVE AGP-SECURITY-CD     TO CRT-COMPANY.
MOVE ACV-ACTIVITY        TO CRT-ACCT-UNIT.
PERFORM 900-ACCT-UNIT-SECURED.
IF (CRT-ACCT-UNIT-SECURED  = "Y")
    MOVE 101                        TO CRT-ERROR-NBR
    MOVE "ACSC"                     TO CRT-ERROR-CAT
```

900-ACCT-UNIT-SEC-SYS

900-ACCT-UNIT-SEC-SYS verifies that the current user has security access to this accounting unit. If the current user is secured from this accounting unit in the specified company and system code, **CRT-ACCT-UNIT-SECURED** is set to Y; otherwise, it is set to N.

For a discussion of security processing, see *Lawson Administration: User Setup and Security*.

Library	DATABASE

Input	**CRT-COMPANY** **CRT-ACCT-UNIT** **CRT-SYSTEM-CODE**
Output	**CRT-ACCT-UNIT-SECURED** (always set)

Programming Example

```
MOVE AGP-SECURITY-CD     TO CRT-COMPANY.
MOVE ACV-ACTIVITY        TO CRT-ACCT-UNIT.
MOVE "AC"                TO CRT-SYSTEM-CODE
PERFORM 900-ACCT-UNIT-SEC-SYS.
IF (CRT-ACCT-UNIT-SECURED  = "Y")
    MOVE 101                        TO CRT-ERROR-NBR
    MOVE "ACSC"                     TO CRT-ERROR-CAT
```

900-COMPANY-SECURED

900-COMPANY-SECURED verifies that the current user has security access to the company. **CRT-COMPANY-SECURED** is set to Y if the current user is secured from this company, otherwise, it is set to N.

For a discussion of security processing, see *Lawson Administration: User Setup and Security*.

Library	DATABASE
Input	**CRT-COMPANY**
Output	**CRT-COMPANY-SECURED** (always set)

Programming Example

```
MOVE APM-COMPANY            TO CRT-COMPANY.

PERFORM 900-COMPANY-SECURED.

IF (CRT-COMPANY-SECURED = "Y")
    GO TO 490-NEXT.
```

900-COMPANY-SEC-SYS

900-COMPANY-SEC-SYS verifies that the current user has security access to the company and system code. **CRT-COMPANY-SECURED** is set to Y if the current user is secured from this company and system code, otherwise, it is set to N.

For a discussion of security processing, see *Lawson Administration: User Setup and Security*.

Library	DATABASE

Input	**CRT-COMPANY**	
	CRT-SYSTEM-CODE	
Output	**CRT-COMPANY-SECURED** (always set)	

Programming Example

```
 MOVE APM-COMPANY            TO CRT-COMPANY.
MOVE "AP"                   TO CRT-SYSTEM-CODE
PERFORM 900-COMPANY-SEC-SYS.

IF (CRT-COMPANY-SECURED = "Y")
    GO TO 490-NEXT.
```

900-IS-HR-EMP-SECURED

900-IS-HR-EMP-SECURED verifies that the current user has security access to a given employee record. The routine sets **WSSEC-SECURED** to Y if the information is secured, meaning the user is *not* authorized to see an employee record; otherwise, it sets **WSSEC-SECURED** to N. The routine also checks process level security based on the values passed in **CRT-COMPANY** and **CRT-PROCESS-LEVEL**.

For a discussion of security processing, see the *Lawson Administration: User Setup and Security* and the *Human Resources User Guide*.

Library	DATABASE	
Input	**WSSEC-SEC-CLASS**	Employee security class
	WSSEC-SEC-LABEL (includes):	
	WSSEC-SEC-LVL	Employee security level
	WSSEC-SEC-LOCATION	Employee security location
	CRT-COMPANY	
	CRT-PROCESS-LEVEL	
Output	**WSSEC-SECURED** (always set)	

900-PROC-LEV-SECURED

900-PROC-LEV-SECURED verifies that the current user has security access to the process level. **CRT-PROC-LEV-SECURED** is set to Y if the current user is secured from access to the process level in the specified company; otherwise, it is set to N.

For a discussion of security processing, see the *Lawson Administration: User Setup and Security*.

Library	DATABASE
Input	**CRT-COMPANY**
	CRT-PROCESS-LEVEL

Output	**CRT-PROC-LEV-SECURED** (always set)

Programming Example

```
MOVE IAC-FROM-TO-CMPY        TO CRT-COMPANY
MOVE IAC-FROM-TO-LOC         TO CRT-PROCESS-LEVEL
MOVE "N"                     TO CRT-PROC-LEV-SECURED
PERFORM 900-PROC-LEV-SECURED
IF (CRT-PROC-LEV-SECURED = "Y")
    MOVE "IC22"             TO CRT-ERROR-CAT
    MOVE 112                TO CRT-ERROR-NBR
    MOVE IC22F1-IAC-FROM-TO-CMPY-FN  TO CRT-FIELD-NBR
    MOVE IAC-FROM-TO-CMPY
                            TO IC22F1-IAC-FROM-TO-CMPY
    MOVE IAC-FROM-TO-LOC
                            TO IC22F1-IAC-FROM-TO-LOC
```

900-PROC-LEV-SEC-SYS

900-PROC-LEV-SEC-SYS verifies that the current user has security access to the process level and system code. **CRT-PROC-LEV-SECURED** is set to Y if the current user is secured from access to the process level in the specified company and for the specified system code; otherwise, it is set to N.

For a discussion of security processing, see the *Lawson Administration: User Setup and Security*.

Library	DATABASE
Input	**CRT-COMPANY**
	CRT-PROCESS-LEVEL
	CRT-SYSTEM-CODE
Output	**CRT-PROC-LEV-SECURED** (always set)

Programming Example

```
MOVE IAC-FROM-TO-CMPY        TO CRT-COMPANY
MOVE IAC-FROM-TO-LOC         TO CRT-PROCESS-LEVEL
MOVE "IC"                    TO CRT-SYSTEM-CODE
MOVE "N"                     TO CRT-PROC-LEV-SECURED
PERFORM 900-PROC-LEV-SEC-SYS
IF (CRT-PROC-LEV-SECURED = "Y")
    MOVE "IC22"             TO CRT-ERROR-CAT
    MOVE 112                TO CRT-ERROR-NBR
    MOVE IC22F1-IAC-FROM-TO-CMPY-FN  TO CRT-FIELD-NBR
    MOVE IAC-FROM-TO-CMPY
                            TO IC22F1-IAC-FROM-TO-CMPY
    MOVE IAC-FROM-TO-LOC
                            TO IC22F1-IAC-FROM-TO-LOC
```

String Processing

String processing services provide a variety of date-to-string or number-to-string conversion and string manipulation functions.

Routines described in "Date and Time Processing" on page 55 are also related to string processing.

500-STRING-ALPHA

500-STRING-ALPHA skips leading and trailing spaces when concatenating **WS-STRFLD-IN** to **WS-STRFLD-OUT**. No error indication is given when the results are truncated.

Library	STRRTNS

Input	**WS-STRFLD-IN**
	WS-STRFLD-IN-LIT-LEN (optional)
	WS-STRFLD-PARENS-FLAG
	WS-STRFLD-PERCENT-FLAG
	WS-STRFLD-PNTR
Output	**WS-STRFLD-IN-GRP** (all input control fields are reinitialized)
	WS-STRFLD-OUT (always set)

Programming Example

```
*      Set negative amount in parentheses. No zero suppress.
       MOVE 1                        TO WS-STRFLD-PNTR.
       MOVE negative-amount          TO WS-STRFLD-IN.
       MOVE 15                       TO WS-STRFLD-IN-LIT-LEN.
       SET STRFLD-NEEDS-PARENS       TO TRUE.
       PERFORM 500-STRING-ALPHA.
```

500-STRING-FIELD

500-STRING-FIELD finds the beginning and ending of an input string by ignoring leading and trailing blanks; adds parentheses, a percent sign, or both, based on input parameter switches set at call; and concatenates the processed string from **WS-STRFLD-IN** into **WS-STRFLD-OUT**. If **WS-STRFLD-IS-ALPHA** is set to Y, leading zeros are not removed. No error indication is given when the results are truncated.

Library	STRRTNS
Input	**WS-STRFLD-IN**
	WS-STRFLD-IN-LIT-LEN (optional)
	WS-STRFLD-IS-ALPHA (optional)
	WS-STRFLD-PARENS-FLAG
	WS-STRFLD-PERCENT-FLAG
	WS-STRFLD-PNTR
Output	**WS-STRFLD-IN-GRP** (all input control fields are reinitialized)
	WS-STRFLD-OUT(always set)

Programming Example

```
*      Set negative amount in parentheses.
       MOVE 1                        TO WS-STRFLD-PNTR.
       MOVE negative-amount          TO WS-STRFLD-IN.
       MOVE 15                       TO WS-STRFLD-IN-LIT-LEN.
       SET STRFLD-NEEDS-PARENS       TO TRUE.
       PERFORM 500-STRING-FIELD.
```

500-STRING-NUMERIC

500-STRING-NUMERIC finds the beginning of an input string by ignoring leading zeros; adds parentheses, a percent sign, or both, based on switches set at call; and concatenates the processed string into **WS-STRFLD-OUT**. No error indication is given when the results are truncated.

Library	STRRTNS
Input	**WS-STRFLD-IN**
	WS-STRFLD-IN-LIT-LEN (optional)
	WS-STRFLD-IS-ALPHA (optional)
	WS-STRFLD-PARENS-FLAG
	WS-STRFLD-PERCENT-FLAG
	WS-STRFLD-PNTR
Output	**WS-STRFLD-IN-GRP** (all input control fields are reinitialized)
	WS-STRFLD-OUT (always set)

Programming Example

```
*      Set negative amount in parentheses with zero suppress.
       MOVE 1                          TO WS-STRFLD-PNTR.
       MOVE negative-amount            TO WS-STRFLD-IN.
       MOVE 15                         TO WS-STRFLD-IN-LIT-LEN.
       SET STRFLD-NEEDS-PARENS         TO TRUE.
       PERFORM 500-STRING-NUMERIC.
```

510-START-STRING

510-START-STRING re-initializes the string area input control fields and resets the result area to spaces.

Library	STRRTNS
Input	No input required.
Output	**WS-STRFLD-IN-GRP** (all input control fields are reinitialized)
	WS-STRFLD-OUT (reset to spaces)

Programming Example

```
       PERFORM 510-START-STRING.
       MOVE GDT-ACCOUNT-DESC       TO WS-STRFLD-IN.
       PERFORM 500-STRING-ALPHA.
       MOVE WS-STRFLD-OUT          TO GDT-SHORT-DESC.
```

520-STRING-LITERAL

520-STRING-LITERAL strings the input area, character by character, to the output area for whichever is shorter: input length, maximum input length, or maximum output length.

Library	STRRTNS
Input	**WS-STRFLD-IN**
	WS-STRFLD-IN-LIT-LEN
Output	**WS-STRFLD-IN-GRP** (all input control fields are reinitialized)
	WS-STRFLD-OUT (with input data added)
	WS-STRFLD-PNTR (set to next available character position in output)

Programming Example

```
PERFORM 510-START-STRING.
MOVE WS-BLK-STRFLD-IN-18        TO WS-STRFLD-IN.
MOVE 18                         TO WS-STRFLD-IN-LIT-LEN.
PERFORM 520-STRING-LITERAL.
```

580-STRING-GROUP

580-STRING-GROUP strings the input area, character by character, to the output area for whichever is shorter: input length, maximum input length, or maximum output length. If the output area is full, no data moves and no error indication is given.

Library	STRRTNS
Input	**WS-STRFLD-IN**
	WS-STRFLD-IN-LIT-LEN
Output	**WS-STRFLD-IN-GRP** (all input control fields initialized)
	WS-STRFLD-OUT (with input data added)
	WS-STRFLD-PNTR (set to next available character position in output)

580-STRING-TELEPHONE

580-STRING-TELEPHONE strings the input data into the output area as (999)999-9999. No error indication is given for an invalid telephone number. If the area code is 0, the number is strung as 999-9999. If the telephone prefix is 0, no data moves to output.

Library	STRRTNS
Input	**WS-STRFMT-TELEPHONE**
Output	**WS-STRFLD-OUT** (optional)
	WS-STRFLD-PNTR (set to next available character position in output)

Programming Example

```
*       Edit the telephone number.
        MOVE telephone-number        TO WS-STRFMT-TELEPHONE.
        PERFORM 580-STRING-TELEPHONE.
```

580-STRING-TIME-HHMMSS

580-STRING-TIME-HHMMSS strings the input data into the output area as HH:MM:SS. If the time value is 0, no data moves to output.

Library	STRRTNS
Input	**WS-STRFMT-TIME**
Output	**WS-STRFLD-OUT** (optional)
	WS-STRFLD-PNTR (set to next available character position in output)

Programming Example

```
MOVE WOH-UPDATE-TIME          TO WS-STRFMT-TIME
MOVE 1                        TO WS-STRFLD-PNTR
MOVE SPACES                   TO WS-STRFLD-OUT
PERFORM 580-STRING-TIME-HHMMSS
MOVE WS-STRFLD-OUT            TO CRT-ERR-VAR3
```

590-COMMON-STRING-FUNCTIONS

The 590-COMMON-STRING-FUNCTIONS section of procedural code has 11 short routines used by the other STRRTNS library routines. These short routines also use the major STRRTNS routines; however, there is no recursive use of code in the STRRTNS library.

590-STRING-COLON

590-STRING-COLON moves a colon to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set)
	WS-STRFLD-PNTR (always set)

590-STRING-DASH

590-STRING-DASH moves a dash to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

590-STRING-DD

590-STRING-DD moves a two-digit day value to the output area without any error checking.

Library	STRRTNS
Input	WS-STRFMT-DATE
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

590-STRING-L-PAREN

590-STRING-L-PAREN moves a left parenthesis to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

590-STRING-MM

590-STRING-MM moves a two-digit month value to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

590-STRING-PERCENT

590-STRING-PERCENT moves a percent sign to the output area without any error checking.

Library	STRRTNS

Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set)
	WS-STRFLD-PNTR (always set)

590-STRING-R-PAREN

590-STRING-R-PAREN moves a right parenthesis to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set)
	WS-STRFLD-PNTR (always set)

590-STRING-SLASH

590-STRING-SLASH moves a forward slash into the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
Output	**WS-STRFLD-OUT** (always set)
	WS-STRFLD-PNTR (always set)

590-STRING-YY

590-STRING-YY moves a two-digit year value preceded by a forward slash to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR**
	WS-STRFMT-DATE
Output	**WS-STRFLD-OUT** (always set)
	WS-STRFLD-PNTR (always set)

590-STRING-YYYY

590-STRING-YYYY moves a four-digit year value preceded by a forward slash to the output area without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR** **WS-STRFMT-DATE**
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

590-STRING-ZM

590-STRING-ZM moves a two-digit month value to the output area and suppresses the leading zeros of months without any error checking.

Library	STRRTNS
Input	**WS-STRFLD-PNTR** **WS-STRFMT-DATE**
Output	**WS-STRFLD-OUT** (always set) **WS-STRFLD-PNTR** (always set)

900-UPPER-TO-LOWER

900-UPPER-TO-LOWER converts text in **WS-CASE-CONV-VALUE** to lowercase. You must provide the character position to start at and the number of characters to convert.

Library	STRRTNS
Input	**WS-CASE-CONV-VALUE** (required) **WS-BEGIN-CONV-AT** (required) **WS-CONV-SIZE** (required)
Output	**WS-CASE-CONV-VALUE**

900-LOWER-TO-UPPER

900-LOWER-TO-UPPER converts text in **WS-CASE-CONV-VALUE** to uppercase. You must provide the character position to start at and the number of characters to convert.

Library	STRRTNS
Input	**WS-CASE-CONV-VALUE** (required) **WS-BEGIN-CONV-AT** (required) **WS-CONV-SIZE** (required)
Output	**WS-CASE-CONV-VALUE**

905-FORMAT-NUMERIC

905-FORMAT-NUMERIC converts a numeric string to the format of the user's locale.

Library	STRRTNS
Input	**WS-NUMBER-IN** (required)
	WS-NUMBER-IS-NEGATIVE (required) (Y or N)
	WSSTR-DECIMAL-SIZE (required)
	CRT-LOCALE (the locale associated with the user)
Output	**WS-NUMBER-OUT** (right justified, zero-suppressed, with a thousand separator and a decimal separator)
	WSDR-ERROR-VAR

Error Number

Invalid Number = 25. This usually means the input was not numeric, included a decimal, or was less than the **WSSTR-DECIMAL-SIZE** value, or the **WSSTR-DECIMAL-SIZE** value was less than zero or greater than nine.

Transaction Processing

Transaction services allow the program to begin and end database transactions.

700-GET-TRANSACTION

700-GET-TRANSACTION gets the transaction waiting at the form that caused the calling program to be scheduled. As the transaction data is passed to the program, many of the common control fields are also reset to point to the current user. Immediately following receipt of the transaction data, the routine gets the current time and date.

Library	CRTRTNS
Input	No input required.
Output	**CRT-TRANSACTION** (always set) CRTWS (system variables set or reset) **WS-DATE-TIME** (always set)

IMPORTANT If a transaction is not available, or if a system error occurs, the END-OF-PROGRAM switch in CRTWS is turned on.

Programming Example

```
*       Get the next transaction.
        PERFORM 700-GET-TRANSACTION.
```

910-AUDIT-BEGIN

910-AUDIT-BEGIN sets the system and database manager in Transaction mode. It defines the start of a unit of work. This routine ensures that the database is up and active.

Library	DATABASE
Input	No input required.
Output	No output returned.

Programming Example

```
IF (HR00F1-FC = "A" OR "C" OR "D")
    PERFORM 910-AUDIT-BEGIN
    IF (DMS-ABORTED)
        GO TO 400-END
    END-IF

    PERFORM 410-UPDATE
    THRU    410-END

    PERFORM 920-AUDIT-END
```

IMPORTANT The caller loses control if the routine finds an error. For example, if the database manager is unavailable, the program is unable to continue.

910-BEGIN-SUB-TRANSACTION

910-BEGIN-SUB-TRANSACTION ensures that a transaction is active and starts a subtransaction of the principal transaction. If a transaction has not been started, the start is forced by this routine.

Library	DATABASE
Input	No input required.
Output	No output returned.

Programming Example

```
PERFORM 900-FIND-BEGRNG-CMT-APCOMMENTS.
IF  (CMTAPC-FOUND)
AND (CMTAPC-NAME         NOT = SPACES)
AND (APC-COMMENT             = SPACES)
AND (AP12F1-LINE-FC (I1) = "A" OR "C")
    PERFORM 910-BEGIN-SUB-TRANSACTION
    MOVE APC-REC-TYPE       TO DB-REC-TYPE
    MOVE APC-COMPANY        TO DB-COMPANY
    MOVE APC-VENDOR         TO DB-VENDOR
    MOVE APC-LOCATION-CODE  TO DB-LOCATION-CODE
    MOVE APC-SEQ-NBR        TO DB-SEQ-NBR
    PERFORM 840-MODIFY-APCSET4
    IF  (APCOMMENTS-FOUND)
        MOVE CMTAPC-NAME    TO APC-COMMENT
        PERFORM 820-STORE-APCOMMENTS
    END-IF
    PERFORM 920-END-SUB-TRANSACTION
END-IF.
```

920-AUDIT-END

920-AUDIT-END ends the current transaction and finishes the current unit of work. This routine also unlocks any database records that the transaction locked. The caller loses control if the routine finds an error.

You must ensure that all programs that update the database include this routine. Updates will not occur if this routine is not present.

Library	DATABASE
Input	No input required.
Output	No output returned.

Programming Example

```
IF (HR00F1-FC = "A" OR "C" OR "D")
    PERFORM 910-AUDIT-BEGIN
    IF (DMS-ABORTED)
        GO TO 400-END
    END-IF

    PERFORM 410-UPDATE
    THRU    410-END

    PERFORM 920-AUDIT-END
```

IMPORTANT The precompiler checks that all online programs that update a database file use this routine.

920-END-SUB-TRANSACTION

920-END-SUB-TRANSACTION validates that the system is in Subtransaction mode and, if so, ends it. If not in Subtransaction mode, this routine ends the current transaction and unit of work and unlocks any locked database records.

Library	DATABASE
Input	No input required.
Output	No output returned.

Programming Example

```
PERFORM 900-FIND-BEGRNG-CMT-APCOMMENTS.
IF  (CMTAPC-FOUND)
AND (CMTAPC-NAME         NOT = SPACES)
AND (APC-COMMENT             = SPACES)
AND (AP12F1-LINE-FC (I1) = "A" OR "C")
    PERFORM 910-BEGIN-SUB-TRANSACTION
    MOVE APC-REC-TYPE       TO DB-REC-TYPE
    MOVE APC-COMPANY        TO DB-COMPANY
    MOVE APC-VENDOR         TO DB-VENDOR
    MOVE APC-LOCATION-CODE  TO DB-LOCATION-CODE
    MOVE APC-SEQ-NBR        TO DB-SEQ-NBR
    PERFORM 840-MODIFY-APCSET4
    IF  (APCOMMENTS-FOUND)
        MOVE CMTAPC-NAME    TO APC-COMMENT
        PERFORM 820-STORE-APCOMMENTS
    END-IF
    PERFORM 920-END-SUB-TRANSACTION
END-IF.
```

Chapter 4

Date and Time Processing

In the following routines, date and time processing issues are handled.

Date and Time Processing

The following date processing routines provide a variety of date and time processing options.

Century Parameter Processing

Dates containing a century of 00 are processed by the date and time routines with a century of 19 or 20, as determined by the century parameter in the configuration file in the univ.cfg file located in the **%LAWDIR%\system** directory. However, most routines ***do not change the century field in the input date***. Only 500-DATE-COMPARE, 501-CHECK-LEAP-YEAR, 510-DATE-EDIT, 520-COMPUTE-JULIAN, and 900-IS-DATE-INVALID replace a century of 00 by storing the century in the input data. For more information, see *Lawson Administration: Server Setup and Maintenance*.

IMPORTANT The **Julian date** is based on the number of days since October 15, 1582.

Deprecated Date and Time APIs

Some older date and time APIs are no longer supported for new development. Documentation for these APIs is provided for backward compatibility only. For more information, see "Deprecated Date and Time Processing" on page 215

900-DATE-ON-CALENDAR

900-DATE-ON-CALENDAR checks the input date against a named calendar to see if the date is marked. The routine returns true (T) if the date is marked; otherwise, it returns false (F). See "Century Parameter Processing" on page 56

NOTE All input dates are YYYYMMDD.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-CALENDAR	A 15	Name of calendar to check.
WSDR-FR-DATE	N 8	YYYYMMDD Date to check as marked on the calendar.

Output

Field	Type and length	Definition
WSDR-DATE-IS-ON-CAL	A 1	Set to Y if input date is marked, otherwise set to N.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Error Number

Error	Explanation
0	No error occurred.
23	Invalid date.
24	Invalid from date.
25	Invalid to date.
26	Invalid calendar.
33	Year Increment Out of Range.
34	Month Increment Out of Range.
35	Day Increment Out of Range.

Programming Example

```
MOVE WHICL-CALENDAR                TO WSDR-CALENDAR.
MOVE WO30F1-WOR-START-DATE         TO WSDR-FR-DATE.
PERFORM 900-DATE-ON-CALENDAR.
IF (WSDR-DATE-IS-ON-CAL NOT = "Y")
    MOVE 250                       TO CRT-ERROR-NBR
    MOVE WO30F1-WOR-START-DATE-FN  TO CRT-FIELD-NBR
```

900-DATE-TO-JULIAN

900-DATE-TO-JULIAN converts the input date to its Julian date number for the input year.

NOTE The input date must be a valid date.

See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to be converted Julian date.

Output

Field	Type and length	Definition
WSDR-JULIAN-DAYS	N 8	Number of days. Returns the resulting JulianDay.
WSDR-ERROR-NBR	N 9	Error number See "Error Number" on page 57 **NOTE** Be sure to set this to zero before using this API. If it is non-zero, **WSDR-JULIAN-DAYS** will be undefined.
WSDR-ERROR-VAR	A 20	Input parameter is in error. Returns the invalid date.

Programming Example

```
IF (PRM-AGING-DAYS          NOT = ZEROES)
    MOVE PRM-CUTOFF-DATE        TO WSDR-FR-DATE
    PERFORM 900-DATE-TO-JULIAN
    SUBTRACT PRM-AGING-DAYS     FROM WSDR-JULIAN-DAYS
    PERFORM 900-JULIAN-TO-DATE
    MOVE WSDR-FR-DATE           TO AP130WS-RECON-DATE
```

900-DAY-FROM-DATE

900-DAY-FROM-DATE calculates the day of the week for the input date. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to be converted.

Output

Field	Type and length	Definition
WSDR-WEEKDAY-NBR	N 8	1 = Sunday, 2 = Monday, and so on.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
MOVE IYR-FYR-FR-DATE        TO WSDR-FR-DATE.
MOVE IYR-FYR-TH-DATE        TO WSDR-TO-DATE.
PERFORM 900-NBR-DAYS-IN-DATE-RNG.
ADD 1                       TO WSDR-NBR-DAYS.

INITIALIZE                  IA80WS-OFFSET-DAYS.
IF (IA80F1-STARTING-WKDAY   NOT = ZERO)
    PERFORM 900-DAY-FROM-DATE
    IF (WSDR-WEEKDAY-NBR    < IA80F1-STARTING-WKDAY)
        ADD 7               TO WSDR-WEEKDAY-NBR
    END-IF
```

900-GET-CALENDAR-DESC

900-GET-CALENDAR-DESC gets the calendar description associated with a named calendar and returns it to the caller. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-CALENDAR	A 15	Name of calendar.

Output

Field	Type and length	Definition
WSDR-CALENDAR-DESC	A 30	Calendar description or an error message.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
MOVE AR00F1-ACG-CALENDAR        TO WSDR-CALENDAR.
PERFORM 900-GET-CALENDAR-DESC.
IF (WSDR-CALENDAR-DESC = AR00WS-CALENDAR-ERR)
    MOVE 204                            TO CRT-ERROR-NBR
    MOVE AR00F1-ACG-CALENDAR-FN         TO CRT-FIELD-NBR
    GO TO 232-END.
```

900-GET-DATE-EOM

900-GET-DATE-EOM takes an input date and returns a valid output date set to the end of the month. If the "from date" is already the last day of the month, the "to date" is set equal to the "from date," unless the "from date" century equals 00.

Library CALRTNS

Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	From date (yyyymmdd)

NOTE If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
WSDR-TO-DATE	N 8	To date (yyyymmdd) Valid end-of-the-month date.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
IF  (IA80WS-MM-DD           = 0101)
AND (IA80F1-NBR-OF-PERIODS  = 12)
     MOVE ZEROS              TO IA80WS-DAYS-PER-PRD
     MOVE IA80WS-YYYY-MM-DD  TO WSDR-FR-DATE
     PERFORM 900-GET-DATE-EOM
     MOVE WSDR-TO-DATE       TO IA80WS-YYYY-MM-DD
```

900-GET-DATE-FROM-NBR-DAYS

900-GET-DATE-FROM-NBR-DAYS gets a result date from a named calendar that is a specified number of marked days from the input date. For an error in calendar name or from date, nothing is returned. The number of calendar days can be negative. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-CALENDAR	A 15	Name of calendar.
WSDR-FR-DATE	N 8	Start date from which to check. If **WSDR-FR-DATE** equals 00000000, then it calculates backwards from the **WSDR-TO-DATE**.
WSDR-NBR-CAL-DAYS	N 8	Specified number of marked days. Always positive.

Output

Field	Type and length	Definition
WSDR-TO-DATE	N 8	Calculated date. When invalid, the field is cleared and left blank. **NOTE** If **WSDR-FR-DATE** = 00000000, then **WSDR-TO-DATE** is the input and **WSDR-FR-DATE** is the output.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
IF (AR25F1-PROCESS-DATE NOT = ZEROS)
    MOVE AR25F1-PROCESS-DATE        TO AR25F1-FROM-DATE
    INITIALIZE AR25F1-PROCESS-DATE
    IF (WS-SV-DUE-TYPE = "M")
        INITIALIZE WSDR-ERROR-NBR
                   WSDR-ERROR-VAR
        MOVE WS-SV-CALENDAR         TO WSDR-CALENDAR
        MOVE AR25F1-FROM-DATE       TO WSDR-FR-DATE
        MOVE WS-SV-DUE-DAYS         TO WSDR-NBR-CAL-DAYS
        PERFORM 900-GET-DATE-FROM-NBR-DAYS
        MOVE WSDR-TO-DATE           TO AR25F1-TO-DATE
```

900-GET-DATE-LIT

900-GET-DATE-LIT converts the date in **WSDR-FR-DATE** to a display format in which the month name is substituted for the number of the month. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to convert.

Output

Field	Type and length	Definition
WSDR-LITERAL-MONTH	A 51	Literal date. When invalid, the field is cleared and left blank.
WSDR-LITERAL-DATE	A 20	Literal month. When invalid, the field is cleared and left blank.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
IF (GLS-FISCAL-YEAR        = PRM-FISCAL-YEAR + 1)
OR (GLS-FISCAL-YEAR        = PRM-FISCAL-YEAR - 1)
OR (GLS-FISCAL-YEAR        = PRM-FISCAL-YEAR)
    MOVE GLS-PER-END-DATE (SUB) TO WSDR-FR-DATE
ELSE
    MOVE GLS-PER-END-DATE (SUB) TO WSDR-FR-DATE
    MOVE PRM-FISCAL-YEAR        TO WSDR-FR-CAL-YEAR.
PERFORM 900-GET-DATE-LIT.
```

900-GET-NBR-DAYS-IN-CAL-LST

900-GET-NBR-DAYS-IN-CAL-LST calculates the number of marked days in a date range from a list of calendars. It only counts the marked days that all of the calendars have in common. The “from” date must precede the “to” date; the order of “from” and “to” is not checked. The number and names of the calendars are validated. The routine returns false (0) if an error is found. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-CAL-LST-GRP	A 15	Contains **WSDR-CAL-LST-GRP** that occurs ten times.
WSDR-NBR-CALENDARS	N 8	Number of calendars.
WSDR-FR-DATE	N 8	Start date for checking marked calendar dates (this date is **not** included).
WSDR-TO-DATE	N 8	End date for checking marked calendar dates (this date **is** included).

Output

Field	Type and length	Definition
WSDR-NBR-DAYS-ON-CAL	N 8	Number of marked days in a date range from the list of calendars.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
*    Find number of available business days.
     MOVE calendar-names               TO WSDR-CAL-LST-GRP.
     MOVE number-of-names              TO WSDR-NBR-CALENDARS.
     MOVE start-date                   TO WSDR-FR-DATE.
     MOVE stop-date                    TO WSDR-TO-DATE.
     PERFORM 900-GET-NBR-DAYS-IN-CAL-LST.
```

900-GET-NBR-DAYS-ON-CAL

900-GET-NBR-DAYS-ON-CAL counts the number of marked days on a specific calendar between a “from” and a “to” date. The “from” date must precede the “to” date; this is not validated. The routine returns false (0) if an error is found. See .

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-CALENDAR	A 15	Name of calendar to check.
WSDR-FR-DATE	N 8	Date to check *from* for marked calendar dates (this date is **not** included).
WSDR-TO-DATE	N 8	Date to check *to* for marked calendar dates (this date **is** included).

Output

Field	Type and length	Definition
WSDR-NBR-DAYS-ON-CAL	N 9	The number of marked days on a specific calendar between the specified dates. Always set.

Field	Type and length	Definition
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
IF (WO20F1-WOH-START-DATE > ZERO)
AND (WO20F1-WOH-END-DATE NOT < WO20F1-WOH-START-DATE)
   MOVE WO20F1-WOH-START-DATE TO WSDR-FR-DATE
   MOVE WO20F1-WOH-END-DATE   TO WSDR-TO-DATE
   PERFORM 900-NBR-DAYS-IN-DATE-RNG

   MOVE WHICL-CALENDAR        TO WSDR-CALENDAR
   PERFORM 900-GET-NBR-DAYS-ON-CAL
   IF (WSDR-NBR-DAYS NOT = WSDR-NBR-DAYS-ON-CAL)
       MOVE 114               TO CRT-MSG-NBR
```

900-GET-WEEKDAY-LIT

900-GET-WEEKDAY-LIT converts the date in **WSDR-FR-DATE** to a display format of the name of the day of the week. Detects the end user's locale and translates the weekday literal for that locale. For an invalid date, the field is cleared and left blank. See "Century Parameter Processing" on page 56

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to convert.

Output

Field	Type and length	Definition
WSDR-LITERAL-WEEKDAY	A 20	Name of the day of the week. This variable returns the full weekday literal, never the abbreviation, in the language determined by the end user's locale. When invalid, the field is cleared and left blank.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
*       Convert date for report heading.
        MOVE report-date                    TO WSDR-FR-DATE.
        PERFORM 900-GET-WEEKDAY-LIT.
```

900-INCREMENT-DATE

900-INCREMENT-DATE takes an input date and processes up to three incremental computations and returns the increment date.

Library CALRTNS

Input

Field	Type and length	Definition
WSDR-FR-DATE	A 8	From date (yyyymmdd). **WSDR-FR-DATE** increments the year and month first to calculate an intermediate date before incrementing the day. The intermediate date must be a valid date. Then, it determines if the original day is still valid in the intermediate month/year. If day is less than 29, it is always valid, but if 29 or greater the intermediate month could be a month with fewer days than the day in the **WSDR-FR-DATE**. If day is greater than the days in the intermediate month, it is adjusted to the last day of the month unless **WSDR-EOM-ROLLOVER** = “Y” in which case the intermediate date is set to the first of the following month. An increment of (0,0,0) returns **WSDR-FR-DATE** equals **WSDR-TO-DATE**, unless century equals 00, in which case the century part of **WSDR-TO-DATE** is set even though it is not set in **WSDR-FR-DATE**.
WSDR-YEAR-INCR	A 5	Year increment can be positive or negative in range of -200 to +200.
WSDR-MONTH-INCR	A 5	Month increment can be positive or negative in range of -2400 to +2400.
WSDR-DAY-INCR	A 5	Day increment can be positive or negative in range of -3200 to +3200.

Field	Type and length	Definition
WSDR-EOM-ROLLOVER	A 1	End of month controls direction when intermediate day is too large. “Y” = roll to first of next month “N” = roll back to end of month **WSDR-EOM-ROLLOVER** only has an effect if the original day exceeds the EOM of the intermediate month/year. This occurs when the day in the **WSDR-FR-DATE** exceeds the numbers of days in the intermediate month. For example, adding one month to August 31 creates an intermediate date in September and the intermediate date must be corrected since September 31 is an invalid date. If **WSDR-EOM-ROLLOVER** equals “N,” it becomes September 30 adding ten days returns October 10. **WSDR-EOM-ROLLOVER** has no effect when the day in the **WSDR-FR-DATE** forms a valid intermediate date.

NOTE If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
WSDR-TO-DATE	A 8	To date (yyyymmdd) Valid date calculated on the increments provided that can be used as input on subsequent calls.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
MOVE WS-PEA-EFFECT-DATE          TO WSDR-FR-DATE.
MOVE ZEROES                      TO WSDR-YEAR-INCR
                                    WSDR-MONTH-INCR.

MOVE 1                           TO WSDR-DAY-INCR.
MOVE "Y"                         TO WSDR-EOM-ROLLOVER.
PERFORM 900-INCREMENT-DATE.
MOVE WSDR-TO-DATE                TO DBRNG-BEG-DATE.
```

900-IS-DATE-INVALID

900-IS-DATE-INVALID validates the input date. The output switch is always set.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Calendar date. **NOTE** If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
WSDR-DATE-ERROR	N 8	False = No error. True = Error.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
MOVE BN70WS-DATE            TO WSDR-FR-DATE.
PERFORM 900-IS-DATE-INVALID.
IF  (WSDR-DATE-ERROR-EXISTS)
     PERFORM 5000-SET-TO-EOM.
```

900-JULIAN-TO-DATE

900-JULIAN-TO-DATE converts a Julian date to a calendar date. The Julian date number is based on the Gregorian start date of October 15, 1582. If the output generated is an invalid date, an error number is returned.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-JULIAN-DAYS	N 8	Julian date.

Output

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Calendar date. **NOTE** If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input in error. Calculated calendar date in error that was calculated from the Julian date input.

Programming Example

```
IF (PRM-AGING-DAYS          NOT = ZEROES)
    MOVE PRM-CUTOFF-DATE        TO WSDR-FR-DATE
    PERFORM 900-DATE-TO-JULIAN
    SUBTRACT PRM-AGING-DAYS     FROM WSDR-JULIAN-DAYS
    PERFORM 900-JULIAN-TO-DATE
    MOVE WSDR-FR-DATE           TO AP130WS-RECON-DATE
```

900-NBR-DAYS-IN-DATE-RNG

900-NBR-DAYS-IN-DATE-RNG calculates the number of calendar days difference between two specified dates. The number of days is always greater than zero.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Calculate from this calendar date: YYYYMMDD **NOTE** If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56
WSDR-TO-DATE	N 8	Calculate to this calendar date: YYYYMMDD

Output

Field	Type and length	Definition
WSDR-NBR-DAYS	N 8	Number of calendar days difference from one date to another. Always positive.
WSDR-ERROR-NBR	N 9	Error number. See "Error Number" on page 57
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
MOVE APP-DUE-DATE           TO WSDR-FR-DATE.
MOVE PRM-PAYMENT-DATE       TO WSDR-TO-DATE.
PERFORM 900-NBR-DAYS-IN-DATE-RNG.
```

905-FORMAT-DATE

The 905-FORMAT-DATE routine formats a date according to the user's locale. This routine can be useful to format dates in messages since the formatting of dates in messages is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to convert (in yyyymmdd format)
CRT-LOCALE	A 30	The user's locale.

Output

Field	Type and length	Definition
WSDR-FORMATTED-DATE	A 51	The date formatted according to the user's locale—for example, if the **WSDR-FR-DATE** is 20031231 and the locale is US English, the output is 12/31/2003.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

905-GET-DATE-LIT

905-GET-DATE-LIT converts the date in **WSDR-FR-DATE** to the date format of the user's locale. In this display format, the month name is substituted for the number of the month. This routine can be useful to format dates in messages since the formatting of dates in messages is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to convert in yyyymmdd format
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Sep rather than September. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-DATE	A 51	Literal date (such as 5 September 2003 or September 5, 2003) An error returns spaces.
WSDR-LITERAL-MONTH	A 20	Literal month An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

905-GET-LOCALE-DATE-LIT

905-GET-LOCALE-DATE-LIT returns the translated month name for the month number in the date in **WSDR-FR-DATE**. The translated name is based on the locale in **WS-LOCALE**. This routine can be useful to format dates in messages since the formatting of dates in messages is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date to convert in yyyymmdd format
WS-LOCALE	A 10	The locale name used to translate the date.
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Sep rather than September. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-DATE	A 51	Literal date (such as 5 September 2003 or September 5, 2003) An error returns spaces.
WSDR-LITERAL-MONTH	A 20	Literal month An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Programming Example

```
*       Convert date for report heading.
        MOVE report-date                    TO WSDR-FR-DATE.
        PERFORM 905-GET-LOCALE-DATE-LIT.
```

905-GET-MONTH-LIT

905-GET-MONTH-LIT returns the translated month name for the month number in the date in **WSDR-FR-DATE**. The translated name is based on the user's locale. This routine can be useful to format months in messages since such formatting is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date, in yyyymmdd format, to use in the conversion of the month name.
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Sep rather than September. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-MONTH	A 20	Literal month An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

905-GET-LOCALE-MONTH-LIT

905-GET-LOCALE-MONTH-LIT returns the translated month name for the month number in the date in **WSDR-FR-DATE**. The translated name is based on the locale in **WS-LOCALE**. This routine can be useful to format months in messages since such formatting is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date, in yyyymmdd format, to use in the conversion of the month name.
WS-LOCALE	A 10	The locale name used to translate the date.
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Sep rather than September. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-MONTH	A 20	Literal month An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

905-GET-WEEKDAY-LIT

905-GET-WEEKDAY-LIT returns the translated day of the week name for the date in **WSDR-FR-DATE**. The translated name is based on the user's locale. This routine can be useful to format days of the week in messages since such formatting is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date, in yyyymmdd format, to use in the conversion of the day of the week name.
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Mon rather than Monday. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-WEEKDAY	A 20	Literal weekday An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

905-GET-LOCALE-WEEKDAY-LIT

905-GET-LOCALE-WEEKDAY-LIT returns the translated day of the week name for the date in **WSDR-FR-DATE**. The translated name is based on the locale in **WS-LOCALE**. This routine can be useful

to format days of the week in messages since such formatting is not done as part of the formatting of other screen elements.

Library CALRTNS

Required Input

Field	Type and length	Definition
WSDR-FR-DATE	N 8	Date, in yyyymmdd format, to use in the conversion of the day of the week name.
WS-LOCALE	A 10	The locale name used to translate the date.
WSDR-DATE-ABBREV-FLAG	A 1	Set to Y to return an abbreviation of the date literal. For example, Mon rather than Monday. The default value is N to return full literal.

Output

Field	Type and length	Definition
WSDR-LITERAL-WEEKDAY	A 20	Literal weekday An error returns spaces.
WSDR-ERROR-NBR	N 9	Error number. 24 = Bad Date in **WSDR-FR-DATE**
WSDR-ERROR-VAR	A 20	Input parameter is in error.

Chapter 5

Batch Processing

Batch job processing routines allow users to create and submit batch jobs using online and batch programs. These application program interfaces can help you manage your batch processing more effectively and efficiently.

Batch Job Processing Considerations

Batch job processing routines allow users to create and submit batch jobs using online and batch programs.

Security Permissions and Batch Job Routines

When one of the batch job routines is called from an online program running though **latm**, the online program is running with **lawson** as the user, rather than the user who is running **lapm**. The **lawson** user might not have the correct security permissions to submit the job. Before using these routines, set the **lawson** user as Security Officer, using Lawson User Security (**laua**). For instructions on Lawson User Security, see *Lawson Administration: User Setup and Security*.

Configuring Security for the Lawson User

STEPS **To configure security for the lawson user**

1. Using a text editor, open the following file:

   `$LAWDIR/system/univ.cfg`

   Note the setting for LAUAMINUID.
2. In the **univ.cfg** file, set LAUAMINUID to 80 for the **lawson** user ID.
3. Save and close the **univ.cfg** file.
4. Access the Lawson User Security (**laua**) utility and define the **lawson** user as a Security Officer.

   For instructions, see *Lawson Administration: User Setup and Security*.
5. Using a text editor, open the **univ.cfg** file and reset LAUAMINUID to its original setting.

Job Creation Processing

900-CREATE-JOB

900-CREATE-JOB creates a job definition, updates the job parameters of an existing job, or both. The entire job, including every step and the parameters for every step, must be defined in the **JOBREQ** group.

STEPS **To create a job using the 900-CREATE-JOB routine**

1. Populate all of the appropriate fields for the **JOBREQ** group.
2. Populate the working storage parameter fields.
3. Move the entire parameter working storage to the **JOBSTP-PARAMS** field.
4. Perform **900-CREATE-JOB** routine.

STEPS **To update job parameters using the 900-CREATE-JOB routine**

1. Use 900-LOAD-JOB to populate the **JOBREQ** group.
2. Modify the necessary fields in the **JOBREQ** group.
3. Perform 900-CREATE-JOB .

For an existing job, the 900-CREATE-JOB routine modifies the steps defined in the **JOBREQ** group and leaves any other pre-existing steps unchanged. For example, if an existing job is defined with six steps in it, and 900-CREATE-JOB is called with a four-step job definition, then the first four steps of the existing job are replaced with the information defined in the four **JOBREQ-STEPS** occurrences, and the last two steps of the existing job are left unchanged.

Users are encouraged to combine the 900-LOAD-JOB routine and **900-CREATE-JOB** routine to modify existing single- and multiple-step jobs.

Library	JOBRTNS
Output	The batch job is created.

900-CREATE-AND-SUBMIT-JOB

900-CREATE-AND-SUBMIT-JOB creates a job, updates the job parameters, or both; and then it submits the batch job to the Job Scheduler.

STEPS **To use the900-CREATE-AND-SUBMIT-JOB. routine**

1. Populate all of the appropriate fields for the **JOBREQ** group.
2. Populate the working storage parameter fields.
3. Move the entire parameter working storage to the **JOBSTP-PARAMS** field.
4. Perform 900-CREATE-AND-SUBMIT-JOB.

   The calling program has no method of knowing when the job actually starts or completes.

Library	JOBRTNS

Unused Fields	(None)
Output	The batch job is submitted to the Job Scheduler.

Required Fields

Field	Type and length	Definition
JOBREQ-CREATE	A 1	Y = Deletes an existing job definition before creating a new one.
JOBREQ-MODIFY	A 1	Y = Deletes an existing job definition before creating a new one. **NOTE** If neither **JOBREQ-CREATE** nor **JOBREQ-MODIFY** has a value of Y, the routine sets **JOBREQ-MODIFY** to Y.
JOBREQ-DELQJB	A 1	Deletes all Job Scheduler entries for the job except for running jobs. If the job already exists and it is running, then the routine fails and returns Y in the **WSJR-ERROR** field.
JOBREQ-NO-MOD-IF-EXISTS	A 1	Forces the routine not to update the job if it already exists. If the job exists, then the routine returns Y in the **WSJR-ERROR** field.
JOBREQ-STEPS	Occurs 20 times	Group label for each step occurrence. The following fields are subscripted.
JOBSTP-TOKEN	A 10	Case-sensitive form ID to be run in this step.
JOBSTP-PROJECT	A 14	Product line, data area, or data ID name for the application form ID. user and environment form IDs have no product line, so in those cases, move spaces to this field.
JOBSTP-PARAMS	A 2000	The parameters for this step, strung together in contiguous order. Not all steps have parameters.

Optional Fields

Field	Type and length	Definition
JOBREQ-DESCRIPTION	A 30	Description for the job.
JOBSTP-DESCRIPTION	A 30	Description for the job step.
JOBSTP-DIST-GRP	A 10	Distribution groups you have defined in Distribution Group Maintenance.
JOBSTP-PRINTER	A 10	Appropriate printer

Field	Type and length	Definition
JOBSTP-NBR-COPIES	N 2	Number of copies to print if sent directly to a printer. The default is to 1.
JOBSTP-SAVE	A 1	Whether or not to save the job definition after it has completed. The default is to Y.
JOBREQ-JOB-QUEUE	A 5	Job queue you want to submit the job through. (If left blank, the default queue is used.)
JOBREQ-START-DATE	N 8	Date the job should start. (If left blank, the job starts now.)
JOBREQ-START-TIME	N 6	Time the job should start. (If left blank, the job starts now.)

Return Value

Field	Type and length	Definition
WSJR-ERROR	A 1	Y = Call failed. N = Call successful.

Programming Example for Application Form ID

```
IF (IC89F1-FC               = "S")
    MOVE CRT-USER-NAME      TO JOBREQ-USER-NAME
    STRING "IC588"         DELIMITED BY " "
           INTO JOBREQ-JOB-NAME
    PERFORM 900-LOAD-JOB
    IF (WSJR-ERROR          = "Y")
        MOVE "Y"            TO JOBREQ-CREATE
        MOVE "N"            TO JOBREQ-IS-MULT-STEP
        MOVE 1              TO JOBREQ-NBR-STEPS
        MOVE "IC588"        TO JOBSTP-TOKEN (1)
        MOVE WS-DB-PRODUCT-LINE
                            TO JOBSTP-PROJECT (1)
        MOVE 1              TO JOBSTP-NBR-COPIES (1)
        MOVE "Y"            TO JOBSTP-SAVE (1)
        MOVE IC89F1-TIME    TO JOBSTP-PARAMS (1)
    ELSE
        MOVE "N"            TO JOBREQ-CREATE
        MOVE "Y"            TO JOBREQ-MODIFY
        MOVE IC89F1-TIME    TO JOBSTP-PARAMS (1)
    END-IF
    PERFORM 900-CREATE-AND-SUBMIT-JOB
    IF (WSJR-ERROR          NOT = "N")
        MOVE 911            TO CRT-ERROR-NBR.
```

Programming Example for User Form ID

```
************************************************
 1000-DO-REPORT                SECTION 50.
*************************************************************
 1000-START.

   MOVE ZEROS                TO RPT-PAGE-COUNT (MO202-R1).
   MOVE ZEROES               TO DB-COMPANY.
   PERFORM 850-FIND-NLT-COMSET1.
```

```
        MOVE "usrtkn1"             TO JOBSTP-TOKEN(1).
        MOVE SPACES                TO JOBSTP-PROJECT(1).
        MOVE "/lawson/usr/lawusr1/USRCMD1"
        TO JOBSTP-PARAMS(1).
        MOVE "N"                   TO JOBREQ-IS-MULT-STEP.
        MOVE "lawusr1"             TO JOBREQ-USER-NAME.
        MOVE "USERTKN1"            TO JOBREQ-JOB-NAME.
        MOVE 1                     TO JOBREQ-NBR-STEPS.
        MOVE "LAWUSR1"               TO WSJR-USER-NAME.
        MOVE "USERTKN1"           TO WSJR-JOB-NAME.
        PERFORM 900-CREATE-AND-SUBMIT-JOB.
        IF WSJR-ERROR = "Y"
            DISPLAY "REQUEST lawusr1, USRTKN1 DID NOT WORK".
        ELSE
            DISPLAY "REQUEST lawusr1, USRTKN1 WORKED".
   PERFORM 1020-DO-COM-COMPANY
     THRU    1020-END.
        UNTIL (COMPANY-NOTFOUND).

     GO TO 1000-END.

*******************************************************
```

Job Submission Processing

900-SUBMIT-JOB

900-SUBMIT-JOB submits an existing batch job to the Lawson Job Scheduler. 900-SUBMIT-JOB uses the **JOBWS** fields instead of the **JOBREQ** fields because most of the fields in **JOBREQ** are not needed to submit an existing job, and it would be inefficient to pass data that never gets used.

STEPS **To use the 900-SUBMIT-JOB routine**

1. Populate the user name and job name fields.
2. Perform 900-SUBMIT-JOB.

Library	JOBRTNS
Unused Fields	(None)
Output	Submits the batch job to the Job Scheduler

Required Fields

Field	Type and length	Definition
WSJR-USER-NAME	A 10	Case-sensitive user name used to define the job.
WSJR-JOB-NAME	A 10	Actual job name used to define the job.

Optional Fields

Field	Type and length	Definition
WSJR-JOB-QUEUE	A 5	Job queue you want to submit the job through. (If left blank, the default queue is used.)
WSJR-START-DATE	N 8	Date the job should start. (If left blank, the job starts now.)
WSJR-START-TIME	N 6	Time the job should start. (If left blank, the job starts now.)

Return Value

Field	Type and length	Definition
WSJR-ERROR	A 1	Y = Call failed. N = Call successful.

Programming Example

Submit a batch job.

```
MOVE "lawusr1"                TO WSJR-USER-NAME.
MOVE "MO202JOB"               TO WSJR-JOB-NAME.
PERFORM 900-SUBMIT-JOB.
```

900-LOAD-JOB

900-LOAD-JOB provides a practical method for easily modifying an existing job without redefining the whole job over again. By using the 900-LOAD-JOB routine, the calling program does not need to know what the current job definition is all the way down to the current parameter settings.

900-LOAD-JOB populates the **JOBREQ** group from the an existing job definition. The **JOBSTP-PRINTER**, **JOBSTP-DIST-GRP**, **JOBSTP-NBR-COPIES**, and **JOBSTP-SAVE** fields are retrieved from the JOBSTPRPT record for the default print file. In other words, the routine retrieves the field values assigned to the *FormID***.prt** print file.

STEPS **To use the 900-LOAD-JOB routine**

1. Perform 900-LOAD-JOB, which populates the **JOBREQ** group from the an existing job definition.
2. Modify the **JOBREQ** group fields as needed.
3. Perform 900-CREATE-JOB or 900-CREATE-AND-SUBMIT-JOB.

Library	JOBRTNS
Optional Fields	(None)
Output	Updates or adds job entries in the JOBSTP job definition file.

Required Fields

Field	Type and length	Definition
JOBREQ-USER-NAME	A 10	Case-sensitive user name used to define the job.
JOBREQ-JOB-NAME	A 10	Actual job name used to define the job.

Unused Fields

The following fields are not populated by 900-LOAD-JOB. The field values remain unchanged after the call to 900-LOAD-JOB. This means that any field values or default values in these fields prior to the call to 900-LOAD-JOB will be the same after the call.

Field	Type and length	Definition
JOBREQ-CREATE	A 1	Y = Deletes an existing job definition before creating a new one.

Field	Type and length	Definition
JOBREQ-MODIFY	A 1	Y = Deletes an existing job definition before creating a new one. **NOTE** If neither **JOBREQ-CREATE** nor **JOBREQ-MODIFY** has a value of Y, the routine sets **JOBREQ-MODIFY** to Y.
JOBREQ-DELQJB	A 1	Deletes all Job Scheduler entries for the job except for running jobs. If the job already exists and it is running, then the routine fails and returns Y in the **WSJR-ERROR** field.
JOBREQ-NO-MOD-IF-EXISTS	A 1	Forces the routine not to update the job if it already exists. If the job exists, then the routine returns Y in the **WSJR-ERROR** field.
JOBREQ-JOB-QUEUE	A 5	Job queue you want to submit the job through. (If left blank, the default queue is used.)
JOBREQ-START-TIME	N 8	Date the job should start. (If left blank, the job starts now.)
JOBREQ-START-DATE	N 6	Time the job should start. (If left blank, the job starts now.)

Return Value

Field	Type and length	Definition
WSJR-ERROR	A 1	Y = Call failed. N = Call successful. M = Maximum number of steps (20) have been read into **JOBREQ** . The job definition contains more steps but the **JOBREQ** group cannot load them all.

Job Output Processing

Print Files

Every print file for a given step gets its printer, distribution group, number of copies, and save flag values from the **JOBSTP-PRINTER**, **JOBSTP-DIST-GRP**, **JOBSTP-NBR-COPIES**, and **JOBSTP-SAVE** fields. This means that every print file for a step has the same distribution group and printer setup. Currently, the batch job processing routines provide no means of setting up different distribution groups for each print file in a step.

Required Fields

Field	Type and length	Definition
JOBREQ-CREATE	A 1	Y = Deletes an existing job definition before creating a new one.
JOBREQ-MODIFY	A 1	Y = Deletes an existing job definition before creating a new one. **NOTE** If neither **JOBREQ-CREATE** nor **JOBREQ-MODIFY** has a value of Y, the routine sets **JOBREQ-MODIFY** to Y.
JOBREQ-DELQJB	A 1	Deletes all Job Scheduler entries for the job except for running jobs. If the job already exists and it is running, then the routine fails and returns Y in the **WSJR-ERROR** field.
JOBREQ-NO-MOD-IF-EXISTS	A 1	Forces the routine not to update the job if it already exists. If the job exists, then the routine returns Y in the **WSJR-ERROR** field.
JOBREQ-USER-NAME	A 10	Case-sensitive user name used to define the job.
JOBREQ-JOB-NAME	A 10	Actual job name used to define the job.
JOBREQ-IS-MULT-STEP	A 1	Y = Multiple-step job.
JOBREQ-NBR-STEPS	N 2	Number of steps (1-20).
JOBREQ-STEPS	Occurs 20 times	Group label for each step occurrence. The following fields are all subscripted.
JOBSTP-TOKEN	A 10	Case-sensitive form ID to be run in this step.
JOBSTP-PROJECT	A 14	Product line, data area, or data ID name for the application form ID. user and environment form IDs have no product line, so in those cases, move spaces to this field.
JOBSTP-PARAMS	A 2000	The parameters for this step, strung together in contiguous order. Not all steps have parameters.

Optional Fields

Field	Type and length	Definition
JOBREQ-DESCRIPTION	A 30	Description for the job.
JOBSTP-DESCRIPTION	A 30	Description for the job step.
JOBSTP-DIST-GRP	A 10	Distribution groups you have defined in Distribution Group Maintenance.
JOBSTP-PRINTER	A 10	Appropriate printer.
JOBSTP-NBR-COPIES	N 2	Number of copies to print if sent directly to a printer. The default is to 1.
JOBSTP-SAVE	A 1	Whether or not to save the job definition after it has completed. The default is to Y.

Unused Fields

Field	Type and length	Definition
JOBREQ-JOB-QUEUE	A 5	Job queue you want to submit the job through. (If left blank, the default queue is used.)
JOBREQ-START-DATE	N 8	Date the job should start. (If left blank, the job starts now.)
JOBREQ-START-TIME	N 6	Time the job should start. (If left blank, the job starts now.)

Return Value

Field	Type and length	Definition
WSJR-ERROR	A 1	Y = Call failed. N = Call successful.

CKPOINT and Restart

The CKPOINT file provides a location for batch programs to store recovered data. A record is automatically created each time a program is executed, and automatically deleted upon the successful completion of the program. If the program must be restarted, the data stored in the CKPOINT record can be used to determine the point from which the program must resume. The field within the CKPOINT file, named CKP-RESTART-INFO, provides storage for this program-specific data. To delete the CKPOINT file, see *Lawson Administration: Server Setup and Maintenance*.

Save Routines for Batch Print File Recovery on Restart

The following routines save print and work file information and report restart logic used for recovery in the event of a batch program failure. This information enables batch update programs using CKPOINT and restart logic to synchronize the recovery of print and work files to the database recovery and restart.

IMPORTANT These APIs stop any existing print and work files from being initialized and set the amount of file data to be saved from the previous partial program execution.

Placement of Automatic Save Routines

The optimum place to perform 900-SAVE-***-FILES is immediately before 920-AUDIT-END. For more information, see "Transaction Processing" on page 52

Enabling and Disabling Automatic Save Routines

To enable the 900-SAVE-***-FILES at compile time, the PRINTRCVON option must be set by creating the file PRINTRCVON in the directories:

$LAWDIR**/system**

- or -

$LAWDIR**/productline**

Once a program with these recovery routines is compiled, auto recovery is enabled until it is recompiled. The presence of this file in the product line directory enables recovery for all programs in that specific product line. The presence of this file in the system directory enables recovery for all product lines.

You can disable 900-SAVE-***-FILES routines at compile time by removing the PRINTRCVON file or by creating a PRINTRCVOFF file in the directory:

$LAWDIR**/system**

- or -

$LAWDIR**/productline**

Either of these actions in the product line directory disables recovery for all programs in that specific product line. In the system directory, they disable recovery for all product lines.

For more information about PRINTRCVON and PRINTRCVOFF, see “PRINTRCVON and PRINTRCVOFF” in *Lawson Administration: Server Setup and Maintenance*.

900 SAVE-WORK-FILES

Directory	$GENDIR and $LAWDIR
Perfom	900-SAVE-WORK-FILES
Output	Work file recovery data for restarting work file processing in a batch program is stored to a .rcv file.

Programming Example

```
*    Save failed batch program work file data for recovery
*    and restart use.
     PERFORM 900-SAVE-WORK-FILES.
```

900-SAVE-OUTPUT-FILES

Directory	$GENDIR and $LAWDIR
Perform	900-SAVE-OUTPUT-FILES
Output	Print and work file recovery data for restarting report printing and work file processing in a batch program is stored to a .rcv file.

Programming Example

```
*     Save failed batch program print and work file data for
*     recovery and restart use.
    PERFORM 900-SAVE-OUTPUT-FILES.
```

Job Deletion Processing

900-DELETE-JOB

900-DELETE-JOB deletes a batch job.

Library	JOBRTNS
Optional Fields	(None)
Output	The batch job is deleted.

Required Fields

Field	Type and length	Definition
WSJR-USER-NAME	A 10	Case-sensitive user name used to define the job.
WSJR-JOB-NAME	A 10	Actual job name used to define the job.

Unused Fields

Field	Type and length	Definition
WSJR-JOB-QUEUE	A 5	Job queue you want to submit the job through. (If left blank, the default queue is used.)
WSJR-START-DATE	N 8	Date the job should start. (If left blank, the job starts now.)
WSJR-START-TIME	N 6	Time the job should start. (If left blank, the job starts now.)

Return Value

Field	Type and length	Definition
WSJR-ERROR	A 1	Y = Call failed. N = Call successful.

Programming Example

```
*       Delete a batch job definition.
        MOVE lawusr1                TO WSJR-USER-NAME.
        MOVE MO202JOB               TO WSJR-JOB-NAME.
        PERFORM 900-DELETE-JOB.
```

900-SAVE-PRINT-FILES

Directory	$GENDIR and $LAWDIR
Perform	900-SAVE-PRINT-FILES
Output	Print file recovery data for restarting report printing in a batch program is stored to a .rcv file.

Programming Example

```
*    Save failed batch program print file data for
*    recovery and restart use.
    PERFORM 900-SAVE-PRINT-FILES.
```

Chapter 6

Database Input/Output Routines

The routines described in this chapter give you access to standard methods for accessing data from your database.

IMPORTANT Lawson recommends that you always use the database input/output routines provided by Lawson to access your Lawson data. This will ensure that your data remains accurate and accessible by Lawson programs and utilities. On rare occasions, you might choose to use non-Lawson tools to view or report on your Lawson data. You should never update Lawson data using non-Lawson tools. Lawson Software does provide some instructions for accessing the database via non-Lawson tools; however, using third-party products correctly is the responsibility of the user.

WARNING Lawson cannot assume any responsibility for the results (damage of data and/or structure) of improperly using non-Lawson tools. Please refer to the third-party tools documentation when attempting to access Lawson data via any non-Lawson tool.

Database Routines in this Chapter

The sections in this chapter include:

Database Update and Filter Routines

For information on other types of data access APIs, see:

Deprecated Database Routines

The direct inquiry data processing APIs (800-OPEN-<FileName> and 860-READ-<FileName>) are no longer supported for new development. Documentation for these APIs is provided for backward compatibility only. For more information, see "Deprecated APIs" on page 214.

Using Input/Output Routines

This section describes how the database input/output API routines are used in a program.

Populating Key Fields and Indexes

To use data access routines in your source code, you must understand the relationships between key fields and indexes. In your source code, you must populate record retrieval variables (also known as **DB** fields) with key field values from working storage, form fields, or record values. Key fields and indexes allow you to specify the records you need to find and update.

Key Fields

Key fields are a set of field values used to identify a record or group of records. You access all database files either sequentially or with key fields. However, even with sequential access, you must define an initial key field value for the first access.

Indexes

An index is a set of key fields used to identify and access records in a file. To allow for different paths or key fields to a record, a file can have up to 16 indexes.

4GL Index Naming Conventions

In the Database Definition utility (**dbdef**), indexes are named using the following format:

<FilePrefix>SET<n>, where

- <FilePrefix> is a three-letter abbreviation for the file.
- <n> is an integer representing the Index number. The following conventions are often used, although they are not a requirement.
- 1 is used for the primary (first) Index.
- 2 through 16 are used for secondary indexes.

  For example, the primary index on the GLMASTER file is GLMSET1.

Populating Record Retrieval Variables

Before you can find or modify records, you must populate record retrieval variables with key field values from working storage, form fields, or record values. This task is sometimes called initializing the values in the **DB** fields because the record retrieval variables each have the name **DB-<FieldName>**.

The reason for moving the key field values is so that values are passed properly. The index you specify in the call determines the key fields you must use to access a record. The record or range of records you must access also determines the key fields you must use.

The Build Shell (bldsh) utility generates a working storage definition for all required record retrieval variables. The database files and input/output routines used determine which variables are required, and only those variables are generated.

Using a Key Field More Than Once

Each database key field has a corresponding **DB** field. However, if the same key field appears in more than one file definition, there is only one **DB** field used for these key fields when accessing the files. For example, COMPANY is a key field used in many files, but **DB-COMPANY** is defined only once in working storage.

Identifying Key Fields in an Index

Because the key fields available for accessing a record are determined by the index used, you frequently need to know which fields are present in the index. The following set of steps describes one method for identifying the fields in an index.

STEPS **To identify the key fields in an index**

1. At the command line, type

   dbdef

2. In the Product Line field, type or select the product line with the index you want to search.
3. Press **Define (F6)** and choose Files.
4. In the File Name field, type or select the name of the file you want to search.
5. Press **Define (F6)** and choose Indexes.
6. In the Index Name field, type or select the name of the index you want to search.

   The list of key fields in the index displays the key fields used by this index to access the file.

For more information on key fields and indexes, see the *Application Development Workbench*.

Moving a Key Field Value to Retrieve a Record

STEPS **To move key field values**

- Move the identifying key field values to a set of record retrieval variables (**DB** fields) using the following format:

```
MOVE <key-field>              TO DB-<field-name>
```

For example, to find a record in the GLMASTER file, you must set the Company, Acct-Unit, Account, and Sub-Account key field values in the appropriate **DB** fields. If the key values currently reside in your working storage, use the following series of **MOVE** commands to fill the **DB** fields:

```
MOVE GL00WS-COMPANY           TO DB-COMPANY.
MOVE GL00WS-ACCT-UNIT         TO DB-ACCT-UNIT.
MOVE GL00WS-ACCOUNT           TO DB-ACCOUNT.
MOVE GL00WS-SUB-ACCOUNT       TO DB-SUB-ACCOUNT.
```

Writing PERFORM Statements Using Database Routines

Write a PERFORM statement that includes the specific database input/output routine name and, in most cases, the index or file name. (In a few cases, no index or file name is required.) The PERFORM statement in your source code generates the input/output routine you specify when the program is built. Because all database routines are sections, you do not need a THRU clause in the PERFORM statement.

Example of a Perform Statement

```
PERFORM 840-FIND-GLMSET1.
```

Database Inquiry Processing

Find routines only read the data for inquiry, not for update or modification. Use these routines to read data from the database files.

840-FIND-<Index>

Name

840-FIND-<Index>

Description

840-FIND-<Index> reads the single database record whose key fields equal the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields. Because this call looks for a specific record, all **DB** fields must have a value entered before performing the routine.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example shows the call used to retrieve a record from the GLMASTER file.

```
*       Find the company record in GLMASTER.
*
        MOVE WS-COMPANY              TO DB-COMPANY.
        MOVE WS-VAR-LEVELS           TO DB-VAR-LEVELS.
        MOVE WS-ACCT-UNIT            TO DB-ACCT-UNIT.
        MOVE WS-ACCOUNT              TO DB-ACCOUNT.
        MOVE WS-SUB-ACCOUNT          TO DB-SUB-ACCOUNT.
        PERFORM 840-FIND-GLMSET1.
```

850-FIND-NLT-<Index>

Name

850-FIND-NLT-<Index>

Description

850-FIND-NLT-<Index> reads the record whose key field is greater than or equal to the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields.

850-FIND-NLT-<Index>, with its ability to reposition within a cursor, is useful for recovery of batch jobs and continuation after a transaction ends in online programs where the end of the main transaction closes all open cursors.

Using the API in a Program

850-FIND-NLT-<Index> must precede a 860-FIND-NEXT-<Index> or 870-FIND-PREV-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement. 850-FIND-NLT-<Index> does not need to point to an existing record (unlike the 840-FIND-<Index> routine). If you define a generic value for the record search, the routine returns a record and sets the <FileName>-FOUND value for any database record successfully read.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. You do not need to fill all **DB** fields with actual values, because the routine does not need to point to an existing record. Instead, fill those fields that are not required as delimiters with zeros or spaces, depending on whether the fields are numeric or alphanumeric fields. If you need to know whether the record exactly matches the key fields you passed, you must check to see if the record matches the generic values you set.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

This routine precedes the routines: 860-FIND-NEXT-<Index> or 870-FIND-PREV-<Index>, as shown in the examples for "860-FIND-NEXT-<Index>" on page 97, or "870-FIND-PREV-<Index>" on page 98.

860-FIND-NEXT-<Index>

Name

860-FIND-NEXT-<Index>

Description

Once you have defined a position in the file with a successful 850-FIND-NLT-<Index>, use the 860-FIND-NEXT-<Index> routine to process database records sequentially in Index order. This routine reads the record following the current record for this index.

Using the API in a Program

850-FIND-NLT-<Index> must precede a 860-FIND-NEXT-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

This example processes a set of records for a particular subaccount from a specific company account. The first three lines show the key fields being set for the appropriate company, account, and subaccount

combination. The index has two other key fields in it, so lines four and five fill those fields with blank values, because they are not going to be used as delimiters.

Once all the index key fields have been populated, the 850-FIND-NLT-GLMSET1 routine uses the specified index to get the first record defined by the key fields. Four conditions are then defined to see if the retrieved record fits the desired set. If so, the record is processed, and 860-FIND-NEXT-GLMSET1 retrieves the next record for checking and processing. Processing continues until 860-FIND-NEXT-<Index> routine returns a value that violates one of the four checking conditions.

```
***  Process a set of records using FIND-NLT and FIND-NEXT
     MOVE WS-COMPANY               TO DB-COMPANY.
     MOVE WS-ACCOUNT               TO DB-ACCOUNT.
     MOVE WS-SUB-ACCOUNT           TO DB-SUB-ACCOUNT.
     MOVE ZEROS                    TO DB-VAR-LEVELS.
     MOVE SPACES                   TO DB-ACCT-UNIT.
     PERFORM 850-FIND-NLT-GLMSET1.
     PERFORM
         UNTIL     (GLMASTER-NOTFOUND)
         OR        (GLM-COMPANY        NOT = DB-COMPANY)
         OR        (GLM-ACCOUNT        NOT = DB-ACCOUNT)
         OR        (GLM-SUB-ACCOUNT    NOT = DB-SUB-ACCOUNT)
                   PERFORM 999-PROCESS-GLM
                   PERFORM 860-FIND-NEXT-GLMSET1
     END-PERFORM.
```

870-FIND-PREV-<Index>

Name

870-FIND-PREV-<Index>

Description

Once you have defined a position in the file with a successful 850-FIND-NLT-<Index>, use the 870-FIND-PREV-<Index> routine to process database records sequentially in reverse <Index> order. This routine reads the record preceding the current record for this index.

Using the API in a Program

You must precede 870-FIND-PREV-<Index> with 850-FIND-NLT-<Index>.

860-FIND-NEXT-<Index> and 870-FIND-PREV-<Index> each require their own 850-FIND-NLT-<Index> calls to function properly; do not use together after a single 850-FIND-NLT-<Index> call.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.

Field	Description
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

This example shows a way to move backward and forward in a file depending on the action being processed. In this example, the index key to be used is set, the action code is checked, and the routine gets either the next record or the previous record.

Notice that the 850-FIND-NLT-<Index> call is made separately for each type of action code.

```
***   The action codes "N" and "P" are used to indicate that
***   the next or previous record is desired, respectively.
      MOVE BL00F1-OEC-COMPANY     TO DB-COMPANY.

      IF (BL00F1-FC = "N")
          PERFORM 850-FIND-NLT-OECSET1
          IF  (OECOMPANY-FOUND)
          AND (OEC-COMPANY = DB-COMPANY)
              PERFORM 860-FIND-NEXT-OECSET1.

      IF (BL00F1-FC = "P")
          PERFORM 850-FIND-NLT-OECSET1
          PERFORM 870-FIND-PREV-OECSET1.
```

Range Find Data Processing

Use these routines to read a range of data from the database files:

Using Range Find Routines

Range find routines only read the data for inquiry, not for update or modification.

The 850-FIND-BEGRNG-<Index> and 850-FIND-SUBRNG-<Index> are used for initial processing of data and the 850-FIND-MIDRNG-<Index> and 850-FIND-MIDSUBRNG-<Index> are used for repositioning within the initial cursor. To continue processing the cursor, once repositioning is done, the 860-FIND-NXTRNG-<Index> is used.

The 850-FIND-MIDRNG-<Index> and 850-FIND-MIDSUBRNG-<Index>, with their ability to reposition within a cursor, are useful for screen scrolling (page up and page down) within online programs, recovery of batch jobs, and continuation after a transaction end in online programs where the end of the main transaction does a close of all open cursors

Performance Advantages in Using Range Find Routines

The Range Find routines perform similarly to the 850-FIND-NLT-<Index> and 860-FIND-NEXT-<Index> calls. The range find routines are better choices to use in many cases than 850-FIND-NLT-<Index> and 860-FIND-NEXT-<Index>, because the range find routines have some added performance advantages.

When 850-FIND-NLT-<Index> and 860-FIND-NEXT-<Index> calls combine to return a series of records, the database interface defines a range of records in a table. Once the 850-FIND-NLT-<Index> call declares a range and retrieves the first record in the set, the application checks the record to see if it matches the keys passed in the call. If it does, the application continues on and processes the next record as long as a <FileName>-FOUND is returned during each check. Unfortunately, this check condition means that the database still needs to return another record even after the end of the appropriate range of records has been reached.

To accomplish this, the interface drops the last key and opens another range, using it to return a record that fails the check. When the application gets this record, a <FileName>-NOTFOUND results, ending the processing of the records.

The range find calls solve the problem of the second range by placing the check in the interface logic. In this case, once the end of the desired range is reached, the next 860-FIND-NXTRNG-<Index> call causes the interface to return a <FileName>-NOTFOUND to the application. Also, in the case of 850-FIND-BEGRNG-<Index>, 850-FIND-SUBRNG-<Index>, and 860-FIND-NXTRNG-<Index>, you no longer need to fill the extra keys with zeros or spaces. (With 850-FIND-MIDRNG-<Index> and 850-FIND-MIDSUBRNG-<Index>, you still need to initialize the extra keys.)

850-FIND-BEGRNG-<Index>

Name

850-FIND-BEGRNG-<Index>

Description

When you need to process a range of data without interruption, 850-FIND-BEGRNG-<Index> retrieves the first database record in the range.

Using the API in a Program

850-FIND-BEGRNG-<Index> must be paired with 860-FIND-NXTRNG-<Index>.

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to the **WS-DB-BEG-RNG**.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "Programming Examples" on page 104.

850-FIND-SUBRNG-<Index>

Name

850-FIND-SUBRNG-<Index>

Description

850-FIND-SUBRNG-<Index> performs similarly to 850-FIND-BEGRNG-<Index>, but it allows for more precision in defining the range, because it lets you specify beginning and ending values on the last key field defining the range.

Using the API in a Program

The database routines perform an equal condition on all **DB** Index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a "where greater-than and less-than" condition on this index field.

850-FIND-SUBRNG-<Index> must precede a 860-FIND-NXTRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Examples

In the example that uses the 850-FIND-BEGRNG-<Index> call, we only define one value for the Sub Account key field. This means that the 850-FIND-BEGRNG-<Index> and 860-FIND-NXTRNG-<Index> calls get only those records where Company, Account, and Sub-Account exactly match the single key field values set in the **DB** fields.

In this new example, we give two values for the last key field defining the range: a beginning value (in **DB-SUB-ACCOUNT**) and an end value (in **DBRNG-SUB-ACCOUNT**). The 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> calls must still get only those records that exactly match the key field

values set in the **DB** fields for Company and Account. However, this time the returned records no longer need to have a value for Sub-Account that exactly matches the key field value defined in the **DB**field. Instead, the record must have a value for Sub-Account that lies within the range specified by the **DB-SUB-ACCOUNT and DBRNG-SUB-ACCOUNT** fields.

```
***     This call processes a subrange of records.
        MOVE WS-COMPANY             TO DB-COMPANY.
        MOVE WS-ACCOUNT             TO DB-ACCOUNT.
        MOVE WS-FROM-SUB-ACCOUNT    TO DB-SUB-ACCOUNT.
        MOVE WS-THRU-SUB-ACCOUNT    TO DBRNG-SUB-ACCOUNT.
        MOVE GLMSET1-SUB-ACCOUNT    TO WS-DB-SUB-RNG.
        PERFORM 850-FIND-SUBRNG-GLMSET1.
        PERFORM
            UNTIL (GLMASTER-NOTFOUND)
            PERFORM 999-PROCESS-GLM
            PERFORM 860-FIND-NXTRNG-GLMSET1
        END-PERFORM.
```

850-FIND-MIDRNG-<Index>

Name

850-FIND-MIDRNG-<Index>

Description

When you need to process a range of data, starting at a given point in the range, use 850-FIND-MIDRNG-<Index>.

Using the API in a Program

You can use this routine by itself or in conjunction with 850-FIND-BEGRNG-<Index> and 860-FIND-NXTRNG-<Index>.

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to **WS-DB-BEG-RNG**.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Examples

This example uses the 850-FIND-BEGRNG-NAMSET1 routine to locate records with the last name of Smith, then uses the 850-FIND-MIDRNG-NAMSET1 routine to find records with the last name of Smith and first name of Pam.

```
MOVE “Smith”                    TO DB-LASTNAME
MOVE NAMSET1-LASTNAME          TO WS-DB-BEG-RNG
PERFORM 850-FIND-BEGRNG-NAMSET1.
*later...
MOVE “Smith”                    TO DB-LASTNAME
MOVE “Pam”                      TO DB-FIRSTNAME
PERFORM 850-FIND-MIDRNG-NAMSET1.
```

The next example uses the 850-FIND-MIDRNG routine by itself to go directly to record with **WS-DB-BEG-RNG**, and process from that record forward.

```
* Uses 850-FIND-MIDRNG by itself to go directly to record
* within BEGRNG and process from that record forward.
MOVE “Smith”                    TO DB-LASTNAME
MOVE NAMSET1-LASTNAME           TO WS-DB-BEG-RNG
MOVE “Pam”                      TO DB-FIRSTNAME
PERFORM 850-FIND-MIDRNG-NAMSET1.
```

This final example sets the initial range value on **ALT-TYPE** field using the 850-FIND-BEGRNG routine, and then sets the mid-range values from the initial 850-FIND-BEGRNG results. Processing with the 850-FIND-MIDRNG routine starts at the next INVOICE value.

```
* Set up the initial BEGRNG on ALT-TYPE
MOVE SMIDF1-SV-ARH-TRANS-TYPE   TO DB-ALT-TYPE
MOVE SMIDF1-SV-ARH-COMPANY      TO DB-COMPANY
MOVE ARHSET6-ALT-TYPE           TO WS-DB-BEG-RNG
PERFORM 850-FIND-BEGRNG-ARHSET6
* Set up MIDRNG from the initial BEGRNG for screen scrolling,
* start at next INVOICE
IF (SMIDF1-FC = "+")
MOVE SMIDF1-PT-ARH-TRANS-TYPE   TO DB-ALT-TYPE
MOVE SMIDF1-PT-ARH-COMPANY      TO DB-COMPANY
MOVE SMIDF1-PT-ARH-INVOICE      TO DB-INVOICE
MOVE ARHSET6-INVOICE            TO WS-DB-BEG-RNG
PERFORM 850-FIND-MIDRNG-ARHSET6
```

850-FIND-MIDSUBRNG-<Index>

Name

850-FIND-MIDSUBRNG-<Index>

Description

When you need to process a range of data, starting at a given point in a subrange, use 850-FIND-MIDSUBRNG-<Index>.

Using the API in a Program

You can use 850-FIND-MIDSUBRNG-<Index> in conjunction with 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index>.

The database routines perform an equal condition on all **DB** index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a “where greater-than and less-than” condition on this index field.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Fill in the **DB** fields that define the point to position at in the mid-subrange, and also initialize any other **DB** fields that do not define the mid-subrange.
DBSRNG-<KeyFieldName>	Define the beginning value of the last key by moving the appropriate values to the record retrieval field named **DBSRNG-<KeyFieldName>** (starting value).
DBRNG-<KeyFieldName>	Define the ending values of the last key by moving the appropriate values to the record retrieval field named **DBRNG-<KeyFieldName>** (ending value).
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

```
* Set up initial SUBRNG for a range of INVOICES whose values
* are in WS-SUB-BEG and * WS-SUB-END

MOVE SMIDF1-SV-ARH-TRANS-TYPE  TO DB-ALT-TYPE
MOVE SMIDF1-SV-ARH-COMPANY     TO DB-COMPANY
MOVE WS-SUB-BEG                TO DB-INVOICE
MOVE WS-SUB-END                TO DBRNG-INVOICE
MOVE ARHSET6-INVOICE           TO WS-DB-SUB-RNG
PERFORM 850-FIND-SUBRNG-ARHSET6

* Set up MIDSUB from the initial SUBRNG for screen scrolling,
* starting at the next * invoice from the displayed group on
* the screen

IF (SMIDF1-FC = "+")
MOVE SMIDF1-PT-ARH-TRANS-TYPE  TO DB-ALT-TYPE
MOVE SMIDF1-PT-ARH-COMPANY     TO DB-COMPANY
MOVE SMIDF1-PT-ARH-INVOICE     TO DB-INVOICE
MOVE ARHSET6-INVOICE           TO WS-DB-SUB-RNG
MOVE WS-SUB-BEG                TO DBSRNG-INVOICE
MOVE WS-SUB-END                TO DBRNG-INVOICE
PERFORM 850-FIND-MIDSUBRNG-ARHSET6
```

860-FIND-NXTRNG-<Index>

Name

860-FIND-NXTRNG-<Index>

Description

When you need to process a range of data without interruption, use 860-FIND-NXTRNG-<Index>.

Using the API in a Program

You must precede 860-FIND-NXTRNG-<Index> with 850-FIND-BEGRNG-<Index>, 850-FIND-SUBRNG-<Index>, 850-FIND-MIDRNG-<Index>, or 850-FIND-MIDSUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

Comparing this example to the example that uses 850-FIND-BEGRNG-<Index> ("850-FIND-BEGRNG-<Index>" on page 101) and 860-FIND-NEXT-<Index> ("860-FIND-NEXT-<Index>" on page 97) shows two major usage differences:

- You do not need to fill the extra Index key fields with blanks. Instead, the **WS-DB-BEG-RNG** variable identifies the last key field to be used as a delimiter. This restricts the routine to return only those records that have values matching those specified for the key fields.
- You no longer need to use the extra OR checks of the **DB** fields in the **UNTIL** clause. The interface performs the checks.

```
***     This call processes a range of records.
        MOVE WS-COMPANY              TO DB-COMPANY.
        MOVE WS-ACCOUNT              TO DB-ACCOUNT.
        MOVE WS-SUB-ACCOUNT          TO DB-SUB-ACCOUNT.
        MOVE GLMSET1-SUB-ACCOUNT     TO WS-DB-BEG-RNG.
        PERFORM 850-FIND-BEGRNG-GLMSET1.
        PERFORM
            UNTIL (GLMASTER-NOTFOUND)
            PERFORM 999-PROCESS-GLM
            PERFORM 860-FIND-NXTRNG-GLMSET1
        END-PERFORM.
```

870-FIND-PRVRNG-<Index>

Name

870-FIND-PRVRNG-<Index>

Description

When you need to process a range of data without interruption, use 870-FIND-PRVRNG-<Index>.

Using the API in a Program

You must precede 870-FIND-PRVRNG-<Index> with 850-FIND-BEGRNG-<Index>, 850-FIND-SUBRNG-<Index>, 850-FIND-MIDRNG-<Index>, or 850-FIND-MIDSUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Key Find Data Processing

Use these routines to read key field data from database files:

Using Key Find Routines

The Key Find routines work just as the other find and find range routines do, except that the Key Find routines return only the key fields from the index. They do not return entire records from data files. Key find routines only read the data for inquiry, not for update or modification.

These Key Find routines are designed to give a performance advantage when only the key fields are needed. The performance advantage results from the elimination of the need to perform a second read for each find call. For a regular find or range find call, the routine reads the index to locate the record, then reads the table to get the actual record. In the case of a key find, the routine reads the Index without subsequently accessing the table, because the index contains all the requested information (the key fields).

840-KFIND-<Index>

Name

840-KFIND-<Index>

Description

840-KFIND-<Index> inquires on the Index for the key fields that match the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields. This call works as the 840-FIND-<Index> ("840-FIND-<Index>" on page 95) call works, but 840-KFIND-<Index> returns only the keys from the Index.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

850-KFIND-NLT-<Index>

Name

850-KFIND-NLT-<Index>

Description

850-KFIND-NLT-<Index> reads the record whose key field is greater than or equal to the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields, but returns only the keys from the index.

Using the API in a Program

850-KFIND-NLT-<Index> must be paired with a 860-KFIND-NEXT-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the KFIND statement. 850-KFIND-NLT-<Index> does not need to point to an existing record (unlike the 840-FIND-<Index> routine). If you define a generic value for the record search, the routine returns a record and sets the <FileName>-FOUND value for any database record successfully read.
Index key fields	You must populate the index key fields used to locate the record before using this call. You do not need to fill all **DB** fields with actual values, because the routine does not need to point to an existing record. Instead, fill those fields that are not required as delimiters with zeros or spaces, depending on whether the fields are numeric or alphanumeric fields. If you need to know whether the record exactly matches the key fields you passed, you must check to see if the record matches the generic values you set.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "850-FIND-NLT-<Index>" on page 96.

860-KFIND-NEXT-<Index>

Name

860-KFIND-NEXT-<Index>

Description

Once you have defined a position in the file with a successful 850-KFIND-NLT-<Index>, use the 860-KFIND-NEXT-<Index> routine to process database records sequentially in Index order. This routine reads the record following the current record for this index, but returns only the keys from the index.

Using the API in a Program

850-KFIND-NLT-<Index> must be paired with a 860-KFIND-NEXT-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the KFIND statement.
Index key fields	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "860-FIND-NEXT-<Index>" on page 97.

850-KFIND-BEGRNG-<Index>

Name

850-KFIND-BEGRNG-<Index>

Description

When you need to process a range of data without interruption, 850-KFIND-BEGRNG-<Index> retrieves the keys of first database record in the range.

Using the API in a Program

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to the **WS-DB-BEG-RNG**.

850-KFIND-BEGRNG-<Index> must be paired with 860-KFIND-NXTRNG-<Index>.

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 850-KFIND-BEGRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the KFIND statement.
Record retrieval variables (also known as **DB** fields)	Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "850-FIND-BEGRNG-<Index>" on page 101.

850-KFIND-SUBRNG-<Index>

Name

850-KFIND-SUBRNG-<Index>

Description

850-KFIND-SUBRNG-<Index> performs similarly to 850-KFIND-BEGRNG-<Index>, but it allows for more precision in defining the range, because it lets you specify beginning and ending values on the last key field defining the range. The routine inquires on the database records in a range. It accesses and returns only the key fields from an index. It does not access or return whole database records from a file. Otherwise, it works as the normal range find calls work.

The database routines perform an equal condition on all **DB** Index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a "where greater-than and less-than" condition on this index field.

Using the API in a Program

850-KFIND-SUBRNG-<Index> must be paired with a 860-KFIND-NXTRNG-<Index>.

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 850-KFIND-SUBRNG-<Index>.

You must populate the delimiting **DB** fields and **WS-DB-SUB-RNG** before using 850-KFIND-SUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the index.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.

Field	Description
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

860-KFIND-NXTRNG-<Index>

Name

860-KFIND-NXTRNG-<Index>

Description

When you need to process a range of data without interruption, use 860-KFIND-NXTRNG-<Index>.

Using the API in a Program

You must precede 860-KFIND-NXTRNG-<Index> with 850-KFIND-BEGRNG-<Index> or 850-KFIND-SUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "860-FIND-NXTRNG-<Index>" on page 106.

Aggregate Range Data Processing

Aggregate range routines have the advantage of allowing you to specify multiple functions per call. They allow you to perform specific functions on one or more columns for a range of rows. The functions you can perform include

- average value for a column
- maximum value for a column
- minimum value for a column
- sum of a column

Use these aggregate range routines :

For information on deprecated forms of aggregate range routine APIs, see "880-INIT and 880-FIND Aggregate Range APIs" on page 225.

Considerations in Using Aggregate Range Routines

Performance Advantages in Using the Aggregate Range APIs

The aggregate range routines were designed to give you performance advantages over using the 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> calls. When you use the aggregate range routines, you improve processing time by using the power of the relational database management system (RDBMS). Because the RDBMS includes all data retrieval and math functions, using the aggregate range routines eliminates the overhead associated with individual table fetches.

Constraints on Using the Aggregate Range APIs

Only the stored field types CURRENCY, NUMERIC, PERCENT, and SIGNED are supported.

You may only specify one aggregate function, regardless of the number of fields to which it is applied. For example, AVG and SUM cannot be processed in the same request.

Aggregate Overflow APIs

A set of aggregate overflow APIs is available. The aggregate overflow APIs differ from the other aggregate APIs in that they check whether the value returned from the aggregate processing would overflow the field in which it would be stored. If the value is too large, the overflow API does not update the field and instead sets a flag to indicate that an overflow has occurred.

880-INIT-DBAG-<Index> and 880-CALC-DBAG-<Index>

Name

880-INIT-DBAG-<Index>

880-CALC-DBAG-<Index>

Description

Aggregate range routines allow you to perform specific functions on one or more columns for a range of rows:

- average value for a column
- maximum value for a column
- minimum value for a column
- sum of a column

Using the APIs in a Program

You can use the aggregate range routines whenever you would otherwise use 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> to access a set of rows for the purpose of applying average, maximum, minimum, or sum functions against a column.

Use the 880-INIT-DBAG-<Index> routine to initialize an aggregation request.

Use the 880-CALC-DBAG-<Index> routine to process an aggregation request against a data set.

The database routines perform an equal condition on all **DB** Index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a “where greater-than and less-than” condition on this index field.

STEPS **To access database records using the aggregate range routines**

1. Set the **DB-**<KeyField> as you would for any database inquiry routine.
2. Express the ending value for the range key:

   ```
   MOVE identifier TO DBRNG-fieldname.
   ```

3. Identify the range key by moving the symbolic constant:

   ```
   MOVE <IndexName>-<IndexFieldName> TO WS-DB-SUB-RNG.
   ```

4. Initialize the aggregation request. This initializes the Boolean request fields and the return values:

   ```
   PERFORM 880-INIT-DBAG-<Index>.
   ```

5. Set Booleans describing which fields to aggregate and what function to apply.

   ```
   SET DBAVG-<Index>-<FieldName> TO TRUE.
   SET DBMAX-<Index>-<FieldName> TO TRUE.
   SET DBMIN-<Index>-<FieldName> TO TRUE.
   SET DBSUM-<Index>-<FieldName> TO TRUE.
   ```

6. Submit the request:

   ```
   PERFORM 880-CALC-DBAG-<Index>.
   ```

7. Test for function validity by referencing the COUNT variable:

   ```
   IF (DBAG-<Index>-COUNT NOT = ZERO) ...
   ```

8. Un-reference the return values in the fields named DBAG-<Index>-<FieldName>:

   ```
   MOVE DBAG-<Index>-<FieldName> TO WS-RESULT.
   ```

Input Values

Field	Description
<Index>	You must specify the index in the statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-SUB-RNG	Identify the range key by moving the symbolic constant <IndexName>-<IndexFieldName> to **WS-DB-SUB-RNG**
• DBAVG-<Index>-<FieldName> Average • DBMAX-<Index>-<FieldName> Maximum • DBMIN-<Index>-<FieldName> Minimum • DBSUM-<Index>-<FieldName> Sum	Use a set of Boolean fields to identify the columns in the table to be returned and the aggregate function to be applied. These fields are named using the following convention: <Index> corresponds to and must match the Index that is specified in the 880-INIT-DBAG-<Index> and 880-CALC-DBAG-<Index> API calls. <FieldName> identifies the column to be aggregated You can request more than one column in a single API call by using the appropriate Boolean field several times with a different field name each time, and you may specify more than one function for each call.

Return Values

Unlike all other database APIs, the values returned by 880-CALC-DBAG-<Index> are not part of the regular file record area. In fact, nothing related to the aggregate range routines alters the contents of the file record area in any way. All return values are dimensioned to the maximum allowable field size of 18 digits. The number of decimal positions is inherited from the database field. The naming convention for these return value fields is shown in the following table.

Field	Description
DBAG-<Function>-<Index>-<FieldName>	You must code the SET prior to the call. See the following usage example, where <FUNCTION> is AVG, MAX, MIN, or SUM. **SET DB<***FUNCTION***>-<***Index***>-<***FIELD***> TO TRUE**

Programming Example

The following example uses the aggregate range routines to specify the SUM function.

```
***Using aggregate range routines to specify the SUM function

MOVE WS-COMPANY               TO DB-COMPANY.
MOVE WS-ACCOUNT               TO DB-ACCOUNT.
MOVE WS-FROM-SUB-ACCT         TO DB-SUB-ACCOUNT.
MOVE WS-THRU-SUB-ACCT         TO DBRNG-SUB-ACCOUNT.
MOVE GLMSET1-SUB-ACCOUNT      TO WS-DB-SUB-RNG.
PERFORM 880-INIT-DBAG-GLMSET1.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD)      TO TRUE.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD+1)    TO TRUE.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD-1)    TO TRUE.
PERFORM 880-FIND-DBAG-GLMSET1.
IF (DBAG-GLMASTER-COUNT NOT = ZERO)
  MOVE DBAG-SUM-GLMSET1-AMOUNT (WS-PERIOD)     TO WS-CURR-PERIOD-TOTAL
  MOVE DBAG-SUM-GLMSET1-AMOUNT (WS-PERIOD + 1) TO WS-NEXT-PERIOD-TOTAL
  MOVE DBAG-SUM-GLMSET1-AMOUNT (WS-PERIOD - 1) TO WS-PREV-PERIOD-TOTAL.
```

Now, compare the code for the aggregate range routines to the code you would have to write to get the same functionality by using the 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> routines, shown in the following example.

```
*** Using SUBRNG routines to specify the SUM function

MOVE WS-COMPANY                    TO DB-COMPANY.
MOVE WS-ACCOUNT                    TO DB-ACCOUNT.
MOVE WS-FROM-SUB-ACCT              TO DB-SUB-ACCOUNT.
MOVE WS-THRU-SUB-ACCT              TO DBRNG-SUB-ACCOUNT.
MOVE GLMSET1-SUB-ACCOUNT           TO WS-DB-SUB-RNG.
PERFORM 850-FIND-SUBRNG-GLMSET1.
PERFORM UNTIL (GLMASTER-NOTFOUND)
  ADD GLM-AMOUNT (WS-PERIOD)       TO WS-CURR-PERIOD-TOTAL
  ADD GLM-AMOUNT (WS-PERIOD + 1)   TO WS-NEXT-PERIOD-TOTAL
  ADD GLM-AMOUNT (WS-PERIOD - 1)   TO WS-PREV-PERIOD-TOTAL
  PERFORM 860-FIND-NXTRNG-GLMSET1
END-PERFORM.
```

880-CALC-DBAGOF

Name

880-CALC-DBAGOF-<Index>

Description

The 880-CALC-DBAGOF-<Index> API is similar to the 880-CALC-DBAG-<Index> API, with the difference that it provides protection against putting overflow values into the field storing the result returned by the aggregate API. If the aggregate result is too large for the field it would be stored in, a flag is set indicating an overflow occurred. You can then add code to control what the program does in the event of an overflow.

Using the API in a Program

Using the 880-CALC-DBAGOF-<Index> API is similar to using the 880-CALC-DBAG-<Index> API. The difference is that a flag is set in the **WS-DB-AGG-OVERFLOW** field to indicate whether or not an overflow occurred. For more information, see "880-INIT-DBAG-<Index> and 880-CALC-DBAG-<Index>" on page 115.

Input Values

Field	Description
<Index>	You must specify the index in the statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-SUB-RNG	Identify the range key by moving the symbolic constant <IndexName>-<IndexFieldName> to **WS-DB-SUB-RNG**
• DBAVG-<Index>-<FieldName> Average • DBMAX-<Index>-<FieldName> Maximum • DBMIN-<Index>-<FieldName> Minimum • DBSUM-<Index>-<FieldName> Sum	Use a set of Boolean fields to identify the columns in the table to be returned and the aggregate function to be applied. These fields are named using the following convention: <Index> corresponds to and must match the Index that is specified in the 880-INIT-DBAG-<Index> and 880-CALC-DBAGOF-<Index> API calls. <FieldName> identifies the column to be aggregated You can request more than one column in a single API call by using the appropriate Boolean field several times with a different field name each time, and you may specify more than one function for each call.

Return Values

Unlike all other database APIs, the values returned by 880-CALC-DBAGOF-<Index> are not part of the regular file record area. In fact, nothing related to the aggregate range routines alters the contents of the file record area in any way. The return values for the aggregate functions are dimensioned to the maximum allowable field size of 18 digits. The number of decimal positions is inherited from the database field. The naming convention for these return value fields is shown in the following table.

Field	Description
DBAG-<Function>-<Index>-<FieldName>	You must code the SET prior to the call. See the following usage example, where <FUNCTION> is AVG, MAX, MIN, or SUM. **SET DB<*FUNCTION*>-<*Index*>-<*FIELD*> TO TRUE**
WS-DB-AGG-OVERFLOW	The field containing the flag to indicate whether or not an overflow occurred ("1" indicates an overflow, "0" indicates no overflow).

880-FILTER-DBAGOF

Name

880-FILTER-DBAGOF-<Index>

Description

The 880-FILTER-DBAGOF-<Index> API is similar to the 880-CALC-DBAGOF-<Index> API, with the difference that it provides for the use of a filter. As with 880-CALC-DBAGOF-<Index>, if the aggregate result is too large for the field it would be stored in, a flag is set indicating an overflow occurred. You can then add code to control what the program does in the event of an overflow.

Using the API in a Program

Using the 880-FILTER-DBAGOF-<Index> API is similar to using the 880-CALC-DBAG-<Index> API. The difference is the use of a filter and that a flag is set in the **WS-DB-AGG-OVERFLOW** field to indicate whether or not an overflow occurred. For more information on setting up the aggregate function, see "880-INIT-DBAG-<Index> and 880-CALC-DBAG-<Index>" on page 115. For information on setting up the filter, see "How Do the Filter Routines Work?" on page 149 and "Setting the Filter Parameter Value" on page 150.

Input Values

Field	Description
<Index>	You must specify the index in the statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-SUB-RNG	Identify the range key by moving the symbolic constant <IndexName>-<IndexFieldName> to **WS-DB-SUB-RNG**
• DBAVG-<Index>-<FieldName> Average • DBMAX-<Index>-<FieldName> Maximum • DBMIN-<Index>-<FieldName> Minimum • DBSUM-<Index>-<FieldName> Sum	Use a set of Boolean fields to identify the columns in the table to be returned and the aggregate function to be applied. These fields are named using the following convention: <Index> corresponds to and must match the Index that is specified in the 880-INIT-DBAG-<Index> and 880-FILTER-DBAGOF-<Index> API calls. <FieldName> identifies the column to be aggregated You can request more than one column in a single API call by using the appropriate Boolean field several times with a different field name each time, and you may specify more than one function for each call.

Return Values

Unlike all other database APIs, the values returned by 880-FILTER-DBAGOF-<Index> are not part of the regular file record area. In fact, nothing related to the aggregate range routines alters the contents of the file record area in any way. The return values for the aggregate functions are dimensioned to the maximum allowable field size of 18 digits. The number of decimal positions is inherited from the database field. The naming convention for these return value fields is shown in the following table.

Field	Description
DBAG-<Function>-<Index>-<FieldName>	You must code the SET prior to the call. See the following usage example, where <FUNCTION> is AVG, MAX, MIN, or SUM. **SET DB<*FUNCTION*>-<*Index*>-<*FIELD*> TO TRUE**
WS-DB-AGG-OVERFLOW	The field containing the flag to indicate whether or not an overflow occurred ("1" indicates an overflow, "0" indicates no overflow). You must add code to determine what the program does in the event of an overflow.

Chapter 7

Database Update and Deletion Routines

This section describes how to initialize the system for updates or deletions and describes the individual routines. The descriptions of the routines give usage notes and examples, and they are grouped according to their functions.

If you need to inquire on the data without modifying it, see "Database Inquiry Processing" on page 95.

Using Update Routines

The following sections provide background information helpful when using the update routines in this chapter.

Initializing the System for Updates

Before you perform any routines that add, delete, or modify the records in a file, you must initialize the system for update processing by performing the 910-AUDIT-BEGIN routine. Do this once in the initialization code of your program. At the end of program execution, you also need to commit the transactions and close the update processing mode by performing 920-AUDIT-END. If you do not perform this call, any changes made to the database during this transaction state are rolled back. For more information on these two calls, see "910-AUDIT-BEGIN" on page 52, and "920-AUDIT-END" on page 53.

Using Modification Routines

The following set of routines are used to access existing records and ready them for updating. The Modify routines update any values in the record, including any indexes which might have changed due to the updated column values.

With the exception of the 820-UPDATE-<Index> routine, calling one of these routines does not actually modify the record. Instead, it returns the record to the record area fields and sets a switch so that any changes made to the field values are made in the file when you perform an 820-STORE-<FileName> call. Without the 820-STORE-<FileName> call, the changes are not made in the file.

The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Using Range Modification Routines

The modify range routines perform similarly to their predecessors, the 860-MODIFY-NLT-<Index> and 860-MODIFY-NEXT-<Index> calls. The modify range routines are better choices to use in many cases than 850-MODIFY-NLT-<Index> and 860-MODIFY-NEXT-<Index>, because the modify range routines have some added performance advantages.

850-MODIFY-BEGRNG-<Index> and 850-MODIFY-SUBRNG-<Index> are used for initial processing of data and 850-MODIFY-MIDRNG-<Index> and 850-MODIFY-MIDSUBRNG-<Index> are used for repositioning within the initial cursor. To continue processing the cursor, once repositioning is done, the 860-MODIFY-NXTRNG-<Index> is used.

The 850-MODIFY-SUBRNG-<Index> and 850-MODIFY-MIDSUBRNG-<Index> , with their ability to reposition within a cursor, are useful for:

- Screen scrolling (page up and page down) within online programs
- Recovery of batch jobs
- Continuation after a transaction end in online programs where the end of the main transaction does a close of all open cursors

For more information on the differences between the find and modify range calls and the performance advantages you gain through the use of range calls, see "Using Range Find Routines" on page 100.

Record and Index Creation

This section describes the function and requirements of the API routines that create records and indexes.

800-CREATE-<FileName>

Name

800-CREATE-<FileName>

Description

Use 800-CREATE-<FileName> initializes the record area associated with the database file to be updated by setting numeric fields to zero and alphanumeric fields to spaces.

Using the API in a Program

For more information, see "820-STORE-<FileName>" on page 127.

Input Values

Field	Description
<FileName>	You must specify the database file name that you want to create a record in.

Programming Example

For more information, see "820-STORE-<FileName>" on page 127.

800-RECREATE-<FileName>

Name

800-RECREATE-<FileName>

Description

Use 810-RECREATE-<FileName> to make new records based on the values in an existing record. 810-RECREATE-<FileName> populates the record area associated with the database file by making a new record that has fields filled by data that currently exists in the record area associated with database file you refer to in the call to the procedure.

Using the API in a Program

See the Programming Example in this section.

Input Values

Field	Description
<FileName>	You must specify the database file name that you want to create a record in.

Programming Example

The following example shows the **810-RECREATE-<FileName>** call used to add a series of new records that are very similar to each other:

- Use 910-AUDIT-BEGIN to open the transaction state.
- Another routine (5500-MAKE-FIRST-RECORD) is called to add a record in the SALESREP file.
- The subsequent PERFORM loop adds more records with the same information as the first record, with the exception of new salesman numbers and names.
- The 810-RECREATE-SALESREP call adds a new record that has the same field values as the previously defined record.
- Another routine gets the new number and name, which are then moved into the appropriate record area fields.
- The 820-STORE-SALESREP call adds the new record to the file.
- The 920-AUDIT-END call closes the transaction state and commits the transaction.

```
***  This routine adds records for a series of sales reps.
***  All of the fields have the same values to be entered
***  except for SALESMAN and NAME.
*
          PERFORM 910-AUDIT-BEGIN.

          PERFORM 5500-MAKE-FIRST-RECORD.
          PERFORM
              UNTIL (NO-MORE-SALESREPS)
              PERFORM 810-RECREATE-SALESREP
              PERFORM 5550-GET-NEW-SALESREP-INFO
              MOVE WS-NEW-SALESREP           TO SAW-SALESMAN
              MOVE WS-NEW-NAME               TO SAW-NAME
              PERFORM 820-STORE-SALESREP
          END PERFORM.

          PERFORM 920-AUDIT-END.
```

Database Modification Processing

This section describes the function and requirements of the database modification API routines.

820-STORE-<FileName>

Name

820-STORE-<FileName>

Description

820-STORE-<FileName> either adds or updates the current record in the file record area. The choice depends on how you define the record in that area.

If you use either 800-CREATE-<FileName> or 810-RECREATE-<FileName>, the subsequent 820-STORE-<FileName> routine adds a new record to the appropriate file and its indexes. You define new key field values in the file record area before performing 820-STORE-<FileName>.

If you perform a modify routine to read the current database record, the subsequent 820-STORE-<FileName> routine updates the appropriate file with the current record area contents. The 820-STORE-<FileName> routine also updates all changed index entries for the changed record.

Using the API in a Program

- Use the 910-AUDIT-BEGIN routine to open a transaction state.
- Make a new blank record in the database using the 800-CREATE-<FileName> routine.
- Use MOVE statements to populate the fields for this new record. If a MOVE statements is left out, the field it populates remains filled with zeros or spaces. The values placed in the record area determine the key field for the record.
- 820-STORE-<FileName> does not use the record retrieval variables (**DB** fields) to access a particular record. Instead, you must set valid values in the record area associated with the file before performing 820-STORE-<FileName>.
- The 820-STORE-<FileName> call adds the new record to the **SALESREP** file.
- 920-AUDIT-END closes the transaction state, committing the transaction to the database.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.

Programming Example

```
***     Add a new record to the SALESREP file.
        PERFORM 910-AUDIT-BEGIN.

        PERFORM 800-CREATE-SALESREP.
        MOVE WS-COMPANY              TO SAW-COMPANY.
        MOVE WS-SALESMAN             TO SAW-SALESMAN.
        MOVE WS-NAME                 TO SAW-NAME.
        MOVE WS-TERRITORY            TO SAW-TERRITORY.
        MOVE WS-COMM-RATE            TO SAW-COMM-RATE.
        MOVE WS-USED-FL              TO SAW-USED-FL.
        MOVE WS-ACTIVE-STATUS        TO SAW-ACTIVE-STATUS.
        PERFORM 820-STORE-SALESREP.

        PERFORM 920-AUDIT-END.
```

820-UPDATE-<FileName>

Name

820-UPDATE-<FileName>

Description

820-UPDATE-<FileName> directly updates a single existing record in a file. It combines the actions of the 840-MODIFY-<Index> and 820-STORE-<FileName> routines into a single call. Use this routine when you already know that a record exists, such as when you have already inquired on or modified a record in a previous call.

Using the API in a Program

The performance advantage derived from this reduction in the number of calls brings with it a pair of requirements.

You must enter a value in all **DB** fields before performing the routine, because this routine can update the key fields in the index, just as the 840-MODIFY-<Index> routine does.

You must populate all record fields, unlike the 840-MODIFY-<Index> routine. This requirement must be enforced because the 820-UPDATE-<FileName> routine replaces the entire existing record with the contents of all the record fields, even if one or more of them are empty.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.

Field	Description
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example shows the same process as the previous example for the 840-MODIFY-<Index> routine, but using the 820-UPDATE-<FileName> call to alter a record. The record has already been retrieved using a Find call.

```
*      Update an individual record to reflect a change.
*
       PERFORM 910-AUDIT-BEGIN.

       MOVE CT01F1-CTF-NAME        TO DB-NAME.
       MOVE CT01F1-CTF-COMPANY     TO DB-COMPANY.

       PERFORM 500-MOVE-DATA
       THRU    500-END.

       PERFORM 820-UPDATE-CTFILENAME.

       PERFORM 920-AUDIT-END.
```

This example shows the same change being made as in the previous example for the 840-MODIFY-<Index> call. The major differences are that you only use a single call, and you populate the **DB** fields and record fields before calling the 820-UPDATE-<Index> routine.

In this case, you must be certain that the 500-MOVE-DATA fills all of the record fields. If it does not, any unfilled record fields cause the 820-UPDATE-<FileName> call to overwrite the corresponding file fields with the initialization values.

840-MODIFY-<FileName>

Lock Records in a File for Modification

840-MODIFY-<FileName>

Description

840-MODIFY-<Index> reads and locks the record whose key field equals the values you set in the **DB** fields corresponding to the index key fields. Because this call looks for a specific record, you must enter a value in all **DB** fields before performing the routine. The record read remains locked until an 920-AUDIT-END.

Using the API in a Program

1. The 910-AUDIT-BEGIN opens the transaction state.
2. The key fields are entered into the **DB** fields.
3. The 840-MODIFY-CUCSET1 call uses the **DB** fields and the index to return the appropriate record and ready it for updating.
4. The PERFORM 500-MOVE-DATA calls another routine that updates the record fields with the new values.
5. The 820-STORE-CUCODES routine updates the changed record in the file.
6. The 920-AUDIT-END call closes the transaction state and commits the transaction.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example shows the 840-MODIFY-<Index> routine used to modify a record. This particular example assumes that the record has already been retrieved using a Find call.

```
*      Update an individual record to reflect a change.
*
       PERFORM 910-AUDIT-BEGIN.
       IF (DMS-ABORTED)
         GO TO 420-END.

       MOVE CU01F1-CUC-CURRENCY-CODE    TO DB-CURRENCY-CODE.
       PERFORM 840-MODIFY-CUCSET1.

       PERFORM 500-MOVE-DATA
       THRU    500-END.
       PERFORM 820-STORE-CUCODES.

       PERFORM 920-AUDIT-END.
```

860-MODIFY-NEXT-<Index>

Name

860-MODIFY-NEXT-<Index>

Description

This routine sequentially process database records for updating in index order. It reads the record following the current record for this index and prepares it for updating by 820-STORE-<FileName>.

Using the API in a Program

You must precede 860-MODIFY-NEXT-<Index> with 850-MODIFY-NLT-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

850-MODIFY-NLT-<Index>

Name

850-MODIFY-NLT-<Index>

Description

850-MODIFY-NLT-<Index> reads the record whose key field is greater than or equal to the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields. This routine also readies the record for updating by 820-STORE-<FileName>.

Using the API in a Program

850-MODIFY-NLT-<Index> must be paired with 860-MODIFY-NEXT-<Index> or 870-MODIFY-PREV-<Index>

850-MODIFY-NLT-<Index> does not need to point to an existing record (unlike the 840-MODIFY-<Index> routine). If you define a generic value for the record search, the routine returns a record and a <FileName>-FOUND for any database record successfully read.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in. **IMPORTANT** If a key field in the index definition is declared as descending, populate that key field with a maximum value rather than a minimum value.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. You do not need to fill all **DB** fields with actual values, because the routine does not need to point to an existing record. Instead, you fill those fields that are not required as delimiters with zeros or spaces, depending on whether the fields are numeric or alphanumeric fields. If you need to know whether the record exactly matches the key fields you passed, you must check that the record matches the generic values you set.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example shows the use of the modify calls to update a series of records in a file.

```
*   Update a series of SALESREP records for all sales reps
*   in a company to reflect a new commission rate.
*
       PERFORM 910-AUDIT-BEGIN.

       MOVE WS-COMPANY        TO DB-COMPANY.
       MOVE ZEROS             TO DB-SALESMAN.
       PERFORM 850-MODIFYNLT-SAWSET1.
       PERFORM
           UNTIL     (SALESREP-NOTFOUND)
           OR        (SAW-COMPANY       NOT = DB-COMPANY)
           MOVE WS-NEW-COMM-RATE        TO SAW-COMM-RATE
           PERFORM 820-STORE-SALESREP
           PERFORM 860-MODIFY-NEXT-SAWSET1
       END PERFORM.

       PERFORM 920-AUDIT-END.
```

870-MODIFY-PREV-<Index>

Name

870-MODIFY-PREV-<Index>

Description

Once you have defined a position in the file with a successful 850-MODIFY-NLT-<Index>, use 870-MODIFY-PREV-<Index> to process the record previous to the current position. This routine reads the record preceding the current record for this index and prepares it for updating by 820-STORE-<FileName>.

Using the API in a Program

You must precede 870-MODIFY-PREV-<Index> with 850-MODIFY-NLT-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Database Range Modification Processing

850-MODIFY-BEGRNG-<Index>

Name

850-MODIFY-BEGRNG-<Index>

Description

When you need to update a range of data without interruption, the 850-MODIFY-BEGRNG-<Index> routine retrieves the first database record in the range and readies it for updating by an 820-STORE-<FileName> routine.

Using the API in a Program

850-MODIFY-BEGRNG-<Index> must be paired with 860-MODIFY-NXTRNG-<Index>.

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 850-MODIFY-BEGRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

For more information, see "860-MODIFY-NXTRNG-<Index>" on page 138.

850-MODIFY-SUBRNG-<Index>

Name

850-MODIFY-SUBRNG-<Index>

Description

850-MODIFY-SUBRNG-<Index> performs similarly to the 850-MODIFY-BEGRNG-<Index>, but it allows for more precision in defining the range, because it lets you specify beginning and ending values on the last key field defining the range.

Using the API in a Program

850-MODIFY-SUBRNG-<Index> must be paired with 860-MODIFY-NXTRNG-<Index>.

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 850-MODIFY-SUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName>.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

850-MODIFY-MIDRNG-<Index>

Name

850-MODIFY-MIDRNG-<Index>

Description

When you need to update a range of data, starting at a given point in the range, use 850-MODIFY-MIDRNG-<Index>. The 850-MODIFY-MIDRNG-<Index> retrieves the first database record in the specified range and readies it for updating by an 820-STORE-<FileName> routine.

Using the API in a Program

You can use 850-MODIFY-MIDRNG-<Index> in a sequence with 850-FIND-SUBRNG-<Index> and 860-MODIFY-NXTRNG-<Index>

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 850-MODIFY-MIDRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

850-MODIFY-MIDSUBRNG-<Index>

Name

850-MODIFY-MIDSUBRNG-<Index>

Description

850-MODIFY-MIDSUBRNG-<Index> performs similarly to 850-MODIFY-MIDRNG-<Index>, but it allows for more precision in defining the range, because it lets you specify beginning and ending values on the last key field defining the range.

Using the API in a Program

850-MODIFY-MIDSUBRNG-<Index> can be used in conjunction with 850-MODIFY-SUBRNG-<Index> and 860-MODIFY-NXTRNG-<Index>.

The database routines perform an equal condition on all **DB** <Index> fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a “where greater-than and less-than” condition on this index field.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	Fill in the **DB** fields that define the point to position at in the mid-subrange, and also initialize any other **DB** fields that do not define the mid-subrange. Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DBSRNG-<KeyFieldName>** (starting value) and **DBRNG-<KeyFieldName>** (ending value). This defines the range itself.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field **WS-DB-SUB-RNG** in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the index.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

860-MODIFY-NXTRNG-<Index>

Name

860-MODIFY-NXTRNG-<Index>

Description

When you need to update a range of data without interruption, 860-MODIFY-NXTRNG-<Index> retrieves the next database record in a range and prepares it for updating by an 820-STORE-<FileName> call.

Using the API in a Program

You must precede 860-MODIFY-NXTRNG-<Index> with 850-MODIFY-BEGRNG-<Index>, 850-MODIFY-SUBRNG-<Index>, 850-MODIFY-SUBRNG-<Index>, or 850-MODIFY-MIDSUBRNG-<Index>.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Return Values

Field	Description
<FileName>-SW	The Modify routines set the <FileName>-SW switch to indicate the outcome of the read. This routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example shows how to update a series of records in a file.

```
*       Update a range of SALESREP records for all sales reps
*       in a company to reflect a new commission rate.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-COMPANY                 TO DB-COMPANY.
        MOVE SAWSET1-COMPANY            TO WS-DB-BEG-RNG.
        PERFORM 850-MODIFY-NXTRNG-SAWSET1.
        PERFORM
            UNTIL (SALESREP-NOTFOUND)
            MOVE WS-NEW-COMM-RATE       TO SAW-COMM-RATE
            PERFORM 820-STORE-SALESREP
            PERFORM 860-MODIFY-NXTRNG-SAWSET1
        END PERFORM.

        PERFORM 920-AUDIT-END.
```

Comparing this example to the routine that uses 850-MODIFY-NLT-<Index> and 860-MODIFY-NEXT-<Index> shows two major usage differences:

- You do not have to fill the extra index key with blanks. Instead, the **WS-DB-BEG-RNG** variable identifies the last key to be used as a delimiter.
- You no longer need to use the extra OR check on the **DB-**field in the **UNTIL** clause. The interface performs the check.

You can also use a 850-FIND-BEGRNG-<Index>, 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> loop for updates with an 840-MODIFY-<Index>. If your use the find loop versus a modify loop, the 840-MODIFY-<Index> must be a different index than that used on the FIND loop.

Data Deletion Routines

This section describes the function and requirements of the data deletion API routines.

Using Delete Routines

There are two sets of data deletion routines. The first (830-DELETE-<FileName>, 830-DELETERNG-<Index>, and 830-DELETESUBRNG-<Index>) delete only the data you specify when calling the routine.

The second set (830-FULL-DELETE-<filename>, 830-FULL-DELETERNG-<Index>, and 830-FULL-DELETESUBRNG-<Index>) enforce the following delete rules as they are set on the database file definition:

Delete Rule	Description
Delete Ignored	No deletion of related records.
Delete Restrict	Related records must be deleted or moved first by the application.
Delete Cascades	Delete all records in the related table described by the relation.

For more information on data deletion rules, see the *Application Developer's Workbench* guide.

830-DELETE-<FileName>

Name

830-DELETE-<FileName>

Description

830-DELETE-<FileName> deletes the current record in the file record area for the specified <FileName>. You must first read this record using a MODIFY routine before using the 830-DELETE-<FileName> call.

830-DELETE-<FileName> does not access any record retrieval variables (**DB** fields). The record area associated with the file already contains the key field. When this routine deletes a database record, it also deletes all index entries associated with that record.

Using the API in a Program

1. The 910-AUDIT-BEGIN opens the transaction state.

2. The key field values for the record are moved to the **DB** fields, then the 840-MODIFY-<Index> routine reads the record and marks it for change.
3. The 830-DELETE-<FileName> routine deletes the record and its associated index entries.
4. The 920-AUDIT-END routine commits the transaction and closes the transaction state.

Input Values

Field	Description
<FileName>	You must specify the database file that you want to delete.

Programming Example

The following example shows the 830-DELETE-<FileName> routine used to drop a single record from a file.

```
*       Delete a single record from SALESREP.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-COMPANY         TO DB-COMPANY.
        MOVE WS-SALESMAN        TO DB-SALESMAN.
        PERFORM 840-MODIFY-SAWSET1.
        PERFORM 830-DELETE-SALESREP.

        PERFORM 920-AUDIT-END.
```

830-DELETERNG-<Index>

Name

830-DELETERNG-<Index>

Description

830-DELETERNG-<Index> deletes all records in a range. When this routine deletes a range of database records, it also deletes all index entries associated with that range of records.

Using the API in a Program

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 830-DELETERNG-<Index>. In recoverable database environments, 830-DELETERNG-<Index> cannot span more than one transaction state. Deleting a very large range of records in a single block creates a very large journal, or transaction, that might exceed a limit. If a range does not realistically fit in a single transaction, you must use a modify/delete loop, with specific attention paid to controlling the size of the logical transactions.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.

Field	Description
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Programming Example

The following example shows the 830-DELETERNG-<Index> routine used to drop a range of records from a file.

```
*       Delete all sales rep entries from a
*       particular company in the SALESREP file.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-COMPANY         TO DB-COMPANY.
        MOVE SAWSET1-COMPANY    TO WS-DB-BEG-RNG.
        PERFORM 830-DELETERNG-SAWSET1.

        PERFORM 920-AUDIT-END.
```

830-DELETESUBRNG-<Index>

Name

830-DELETESUBRNG-<Index>

Description

830-DELETESUBRNG-<Index> deletes all records in a subrange.

You specify the range similarly to the 830-DELETERNG-<Index> call, but with the addition of another **DB-** field named **DBRNG-<IndexFieldName>**, which is used to specify the end value of the last field in the subrange.

Using the API in a Program

You must populate the delimiting **DB** fields and **WS-DB-SUB-RNG** before using 830-DELETESUBRNG-<Index>.

When this routine deletes a subrange of database records, it also deletes all index entries associated with that subrange of records.

NOTE In recoverable database environments, 830-DELETESUBRNG-<Index> cannot span more than one transaction state. Deleting a very large subrange of records in a single block creates a very large journal, or transaction, that may exceed a limit. If a subrange does not realistically fit in a single transaction, you must use a modify/delete loop, with specific attention paid to controlling the size of the logical transactions.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
DBSRNG-<KeyFieldName> **DBRNG-<KeyFieldName>**	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DBSRNG-<KeyFieldName>** (starting value) and **DBRNG-<KeyFieldName>** (ending value). This defines the range itself.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

830-FULL-DELETE-<FileName>

Name

830-FULL-DELETE-<FileName>

Description

830-FULL-DELETE-<FileName> deletes the current record in the file record area for the specified <FileName>.

When this routine deletes a database record, it also deletes all index entries associated with that record and any related records according to the following delete rules as set in the database file definition.

Delete Rule	Description
Delete Ignored	No deletion of related records.
Delete Restrict	Related records must be deleted or moved first by the application.
Delete Cascades	Delete all records in the related table described by the relation.

For more information on data deletion rules, see the *Application Development Workbench* guide.

You must first read this record using a MODIFY routine before using the 830-FULL-DELETE-<FileName> call.

830-FULL-DELETE-<FileName> does not access any record retrieval variables (**DB** fields). The record area associated with the file already contains the key field.

Using the API in a Program

1. The 910-AUDIT-BEGIN opens the transaction state.
2. The key field values for the record are moved to the **DB** fields, then the 840-MODIFY-<Index> routine reads the record and marks it for change.
3. The 830-FULL-DELETE-<FileName> routine deletes the record and its associated index entries.
4. The 920-AUDIT-END routine commits the transaction and closes the transaction state.

Input Values

Field	Description
<FileName>	You must specify the database file that you want to delete.

Programming Example

The following example shows the 830-FULL-DELETE-<FileName> routine used to drop a single record from a file.

Deleting a single record

```
*       Delete a single record from SALESREP.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-COMPANY         TO DB-COMPANY.
        MOVE WS-SALESMAN        TO DB-SALESMAN.
        PERFORM 840-MODIFY-SAWSET1.
        PERFORM 830-FULL-DELETE-SALESREP.

        PERFORM 920-AUDIT-END.
```

830-FULL-DELETERNG-<Index>

Name

830-FULL-DELETERNG-<Index>

Description

830-FULL-DELETERNG-<Index> deletes all records in a range.

When this routine deletes a database record, it also deletes all index entries associated with that record and any related records according to the following delete rules as set in the database file definition.

Delete Rule	Description
Delete Ignored	No deletion of related records.
Delete Restrict	Related records must be deleted or moved first by the application.
Delete Cascades	Delete all records in the related table described by the relation.

For more information on data deletion rules, see the *Application Development Workbench* guide.

Using the API in a Program

You must populate the delimiting **DB** fields and **WS-DB-BEG-RNG** before using 830-FULL-DELETERNG-<Index>. In recoverable database environments, 830-FULL-DELETERNG-<Index> cannot span more than one transaction state. Deleting a very large range of records in a single block creates a very large journal, or transaction, that might exceed a limit. If a range does not realistically fit in a single transaction, you must use a modify/delete loop, with specific attention paid to controlling the size of the logical transactions.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Programming Example

The following example shows the 830-FULL-DELETERNG-<Index> routine used to drop a range of records from a file.

```
*       Delete all sales rep entries from a
*       particular company in the SALESREP file.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-COMPANY         TO DB-COMPANY.
        MOVE SAWSET1-COMPANY    TO WS-DB-BEG-RNG.
        PERFORM 830-FULL-DELETERNG-SAWSET1.

        PERFORM 920-AUDIT-END.
```

830-FULL-DELETESUBRNG-<Index>

Name

830-FULL-DELETESUBRNG-<Index>

Description

830-FULL-DELETESUBRNG-<Index> deletes all records in a subrange.

When this routine deletes a subrange of database records, it also deletes all index entries associated with that subrange of records and all related records according to the delete rules as set in the database file definition.

Delete Rule	Description
Delete Ignored	No deletion of related records.
Delete Restrict	Related records must be deleted or moved first by the application.
Delete Cascades	Delete all records in the related table described by the relation.

For more information on data deletion rules, see the *Application Development Workbench* guide.

Using the API in a Program

You specify the range similarly to the 830-FULL-DELETERNG-<Index> call, but with the addition of another **DB** field named DBRNG-<IndexFieldName>, which is used to specify the end value of the last field in the subrange.

You must populate the delimiting **DB** fields, the **DBRNG** field, and **WS-DB-SUB-RNG** before using 830-FULL-DELETESUBRNG-<Index>.

NOTE In recoverable database environments, 830-FULL-DELETESUBRNG-<Index> cannot span more than one transaction state. Deleting a very large subrange of records in a single block creates a very large journal, or transaction, that may exceed a limit. If a subrange does not realistically fit in a single transaction, you must use a modify/delete loop, with specific attention paid to controlling the size of the logical transactions.

Input Values

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
DBSRNG-<KeyFieldName> **DBRNG-<KeyFieldName>**	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DBSRNG-<KeyFieldName>** (starting value) and **DBRNG-<KeyFieldName>** (ending value). This defines the range itself.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.

Programming Example

The following example shows the **830-FULL-DELETESUBRNG-<Index>** routine used to drop a subrange of records from the a file.

```
*       Delete all sales reps from a subrange of
*       companies in the SALESREP file.
        PERFORM 910-AUDIT-BEGIN.

        MOVE WS-FROM-COMPANY        TO DB-COMPANY.
        MOVE WS-THRU-COMPANY        TO DBRNG-COMPANY.
        MOVE SAWSET1-COMPANY        TO WS-DB-SUB-RNG.
        PERFORM 830-FULL-DELETESUBRNG-SAWSET1.

        PERFORM 920-AUDIT-END.
```

Chapter 8

Database Index Filter Routines

Using Index Filter Routines

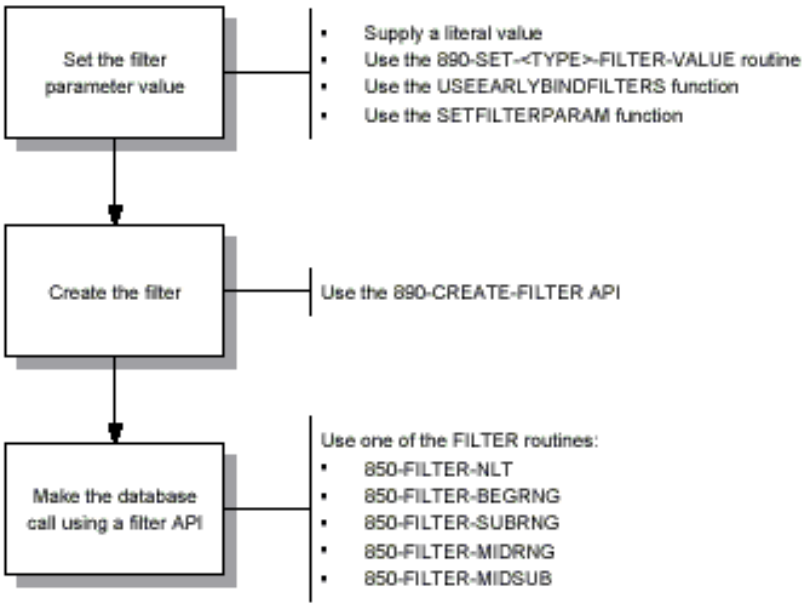
This section describes how filter routines are used.

How Do the Filter Routines Work?

The Index Filter APIs allow creation and setup of a filter. Filters also allow programmers to create conditional statements for the filters. The syntax for the conditional statements allows programmers to filter against literal values and against values entered at run time.

The use of index filters in programming can increase performance, especially for applications that retrieve large amounts of data and must filter that data to find the records that need to be processed.

Figure 1. Process flow: Using the Filter APIs in a Program



Setting the Filter Parameter Value

This section describes how to configure your filter.

Setting the Filter Parameter Value

You can use the following methods to set the value of a filter parameter:

CAUTION For any given filter, you can only use literal values and one of the other methods. Do not mix the other methods in the same filter.

Method 1: Supply a Literal Value

Supply a literal value—for example, (EMP-EMP-STATUS = 'FT').

This example assumes that a table called FILTER with the prefix FLT is set up with two fields. A is an alpha field of length 3 and S is a signed numeric field.

```
MOVE "((FLT-A = ?) OR (FLT-A = 'A  ') AND (FLT-S = ?))" TO
        FILTER-STRING
```

Method 2: Use the Set Filter Value API Routine

Set a value with 890-SET-<Type>-FILTER-VALUE after calling 890-CREATE-FILTER. In this case, the **FILTER-STRING** statement includes the “?” parameter marker syntax. For example:

```
MOVE “(ARO-TRANS-DATE <= ?)”          TO FILTER-STRING
PERFORM 890-CREATE-FILTER.
MOVE AR90F1-SEL-TRANS-DATE-TO   TO DATETIME-FILTER-VALUE.
PERFORM 890-SET-DATETIME-FILTER-VALUE.
```

Method 3: Set the Parameter Marker with the USEEARLYBINDFILTERS Function

For more information about using USEEARLYBINDFILTERS, see "Additional Guidelines for Filters" on page 152 and "Sample Filter" on page 153.

The value of a parameter marker (the ? syntax) can be set before calling 890-CREATE-FILTER if you use the USEEARLYBINDFILTERS function. This function lets the run-time system know that the program will be setting values before the filter is created. This setting remains in effect until you call 890-CREATE-FILTER. 890-CREATE-FILTER changes this setting back to the default behavior.

Call USEEARLYBINDFILTERS and then set a value with 890-SET-<Type>-FILTER-VALUE before calling 890-CREATE-FILTER. For example:

```
CALL "USEEARLYBINDFILTERS" USING WS-TRUE.
IF (AR90F1-SEL-DUE-DATE-FROM    > ZEROS)
    STRING "(ARO-DUE-DATE >= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-FROM    TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
IF (AR90F1-SEL-DUE-DATE-TO      > ZEROS)
    STRING " AND (ARO-DUE-DATE <= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-TO      TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
PERFORM 890-CREATE-FILTER.
```

Method 4: Set the Filter Parameter Value with the SETFILTERPARAM Function

Bind the parameter to a working storage field with SETFILTERPARAM. For example:

```
MOVE "(GAM-COMPANY = ?)" TO FILTER-STRING.
CALL "SETFILTERPARAM" USING GLT-TO-COMPANY.
PERFORM 890-CREATE-FILTER.
```

Index Filter Working Storage Variables

In your program, you will match the appropriate working storage variables with the appropriate API:

```
FILTER-STRING             PIC X(1000)         VALUE SPACES
ALPHANUM FILTER-VALUE     PIC X(100)          VALUE SPACES
NUMERIC-FILTER-VALUE      PIC 9(18)           VALUE ZEROES
SIGNED0-FILTER-VALUE      PIC 9(17)           VALUE ZEROES
SIGNED1-FILTER-VALUE      PIC 9(16)V9(1)      VALUE ZEROES
SIGNED2-FILTER-VALUE      PIC 9(15)V9(2)      VALUE ZEROES
SIGNED3-FILTER-VALUE      PIC 9(14)V9(3)      VALUE ZEROES
SIGNED4-FILTER-VALUE      PIC 9(14)V9(4)      VALUE ZEROES
SIGNED5-FILTER-VALUE      PIC 9(14)V9(5)      VALUE ZEROES
SIGNED6-FILTER-VALUE      PIC 9(14)V9(6)      VALUE ZEROES
SIGNED7-FILTER-VALUE      PIC 9(14)V9(7)      VALUE ZEROES
DATETIME-FILTER-VALUE     PIC 9(8)            VALUE ZEROES
```

Filter String Formatting Rules and Examples

FILTER-STRING is an alpha field that must be parsed by the database filter APIs; therefore the contents must be in a “well-known” format. The criteria for that format are:

- Each condition must start with a left parenthesis “(“ character.

- Each condition must end with a right parenthesis “)” character.
- Valid comparison operators include: =, <, >, <=, >=, <>, !=, LIKE, and NOT LIKE. <> and != are “not equal to.” LIKE and NOT LIKE are used for pattern matching in alpha fields. For pattern matching, you can use the percent sign (%) to represent any character zero or more times, and the underscore (_) to match any character exactly one time. Pattern matching is case sensitive. For example:

  LIKE ‘SM%' matches values of SM, SMITH, SMYTH, SMORES, but not Smith

  LIKE ‘A%C' matches values of AC, ABC, ABDEFC, ACIDIC

  LIKE ‘A_C%' matches values of ABC, ABCDEF, and ARCTIC, but not AC or ACDEFC
- If a literal value is provided, it must be enclosed in single quotes. (The use of double quotes is supported for backwards compatibility, but this use is deprecated and will be removed in a future release.)
- If a literal value is provided, it may only contain A – Z or 0 – 9.
- You cannot use **COBOL** reserved words, such as **ZEROS** or **LOW-VALUES**. If you want the filter to check for **ZEROS**, you need to use either the “?” placeholder or the physical numeric value “0000”.
- If multiple search conditions are provided, they must be connected with either an “AND” or an “OR” conjunction. (You can also place an AND or OR at the beginning of the **FILTER-STRING** without causing an error, but it will be ignored and replaced with an implicit AND.)
- The total count of “(“ and “)” must balance.
- If you include literal values in the filter string, the value's format must exactly match the PIC format of the working storage of the field that you are comparing to. Thus, you might need to space-pad alpha field values or zero-pad numeric field values. For example, if you want to provide a literal value of “AR” for the GLT-SYSTEM field and it is a **PIC A(05)** field, you would format the condition:

  ```
  “(GLT-SYSTEM = ‘AR   ‘)”
  ```

- You can provide more than one value for a single field. This can be done both with ranges of values “((ARO-DUE-DATE >= ?) AND (ARO-DUE-DATE <= ?))” and with lists of values “((GLT-STATUS = '1') OR (GLT-STATUS = '3'))”. The filter string does not support comparisons to multiple values. That is, you cannot say “(GLT-STATUS = '1' OR '3')”.
- To specify an occurring field, use (number). You cannot use a working storage field to specify the value. The literal value must be supplied. That is, “(GAM-DB-AMOUNT(I2) >= ?)” is not valid because the filter APIs have no idea what the value of I2 is, but “(GAM-DB-AMOUNT(10) >= ?)” is valid and will compare the supplied value against the 10th occurrence of DB-AMOUNT.

FILTER-STRING Content Example

```
(EMP-EMP-STATUS = 'FT')
AND (EMP-EMP-STATUS = 'FT')
((EMP-ZIP LIKE '55102%') and ((EMP-EMP-STATUS = 'FT') OR
   (EMP-EMP-STATUS = 'PT')))
((ARO-TRANS-DATE >= ?) AND (ARO-TRANS-DATE <= ?))
AND (ARO-TRANS-DATE >= ?) AND (ARO-TRANS-DATE <= ?)
```

Additional Guidelines for Filters

- You can only specify a database filter for database fields. You cannot specify a database filter for derived fields, string fields, and other non-database fields.
- If you use both “AND” and “OR” conjunctions in a filter string, be sure to use parentheses to control precedence.

- A filter can have no more than 32 separate search conditions.
- The largest field that can be searched (in 9.0.x) is 127 bytes long. This allows all numeric fields to be searched (including BCD, YYYYMMDD, TIME, and so on), but may limit the searching of alpha fields in some cases.
- The total size of fields that can be searched is less than the product of these two limits. That is, you cannot search 32 x 127 byte fields. All of the filter information in the C filter APIs (file handle, field numbers and occurrence numbers, filter description, filter values, and so on) must fit into a 4 KB memory region. If you create a filter that requires more than 4 KB of memory to describe, a run-time error will occur when you call the Index Filter function that uses the filter.
- In addition to any groupings provided in the filter string, the Index Filter APIs will group all of the conditions in a single set of parentheses and the filter will be ANDed with the keys provided to the Index Filter API. So the entire set of search criteria is (keys) AND (filter). An AND or OR conjunction provided at the beginning of the filter string is ignored and replaced by this implicit AND. That is, there is no way to say (keys) OR (filter).
- Filter values should be set in the same place that the filter string is created. This ensures that the logic used to set the values is exactly the same as the logic that was used to create the filter string.
- For every “?” in **FILTER-STRING**, you must call one of the 890-SET-<Type>-FILTER-VALUE functions. The values must be set in the same order that they appear in **FILTER-STRING**.
- The value of a parameter marker (the ? syntax) can be set before calling 890-CREATE-FILTER if you use the USEEARLYBINDFILTERS function. This function lets the run-time system know that the program will be setting values before the filter is created. This setting remains in effect until you call 890-CREATE-FILTER. 890-CREATE-FILTER changes this setting back to the default behavior.
- Calling 890-CREATE-FILTER with spaces in **FILTER-STRING** creates a filter that has no conditions and that can be used with the Index Filter APIs.

Sample Filter

The following code shows a filter that allows a user to enter “to” and “from” values for ARO-DUE-DATE. Because the filter values are set before the call to 890-CREATE-FILTER, the USEEARLYBINDFILTERS function is used.

```
CALL "USEEARLYBINDFILTERS" USING WS-TRUE.
IF (AR90F1-SEL-DUE-DATE-FROM    > ZEROS)
    STRING "(ARO-DUE-DATE >= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-FROM    TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
IF (AR90F1-SEL-DUE-DATE-TO      > ZEROS)
    STRING " AND (ARO-DUE-DATE <= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-TO      TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
PERFORM 890-CREATE-FILTER.
PERFORM 850-FILTERNLT-AROSET5.
```

890-CREATE-FILTER

Name

890-CREATE-FILTER

Description

Creates the filter, based on the value of the **FILTER-STRING** variable.

Using the API in a Program

Before calling the API, set the **FILTER-STRING** variable with a conditional statement.

Use the “?” symbol for any variables to be set at run time.

Input Values

Field	Description
FILTER-STRING	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

None.

Programming Example

The following code shows a filter that allows a user to enter “to” and “from” values for ARO-DUE-DATE. Because the filter values are set before the call to 890-CREATE-FILTER, the USEEARLYBINDFILTERS function is used.

```
CALL "USEEARLYBINDFILTERS" USING WS-TRUE.
IF (AR90F1-SEL-DUE-DATE-FROM    > ZEROS)
    STRING "(ARO-DUE-DATE >= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-FROM    TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
IF (AR90F1-SEL-DUE-DATE-TO      > ZEROS)
    STRING " AND (ARO-DUE-DATE <= ?)" DELIMITED BY SIZE
           INTO FILTER-STRING
           POINTER AR90WS-FILTER-LENGTH
    MOVE AR90F1-SEL-DUE-DATE-TO      TO DATETIME-FILTER-VALUE
    PERFORM 890-SET-DATETIME-FILTER-VALUE.
PERFORM 890-CREATE-FILTER.
PERFORM 850-FILTERNLT-AROSET5.
```

890-SET-<Type>-FILTER-VALUE

Name

The APIs have names built with the type of filter parameter that is being set.

Filter parameter type	API name
Alphanumeric	890-SET-ALPHANUM-FILTER-VALUE
Numeric	890-SET-NUMERIC-FILTER-VALUE

Filter parameter type	API name
Date and time	890-SET-DATETIME-FILTER-VALUE
Signed	There is a separate call for every signed value for the number of decimal places 0 to 7. The total number of digits is limited to 18. • 890-SET-SIGNED-FILTER-VALUE0 • 890-SET-SIGNED-FILTER-VALUE1 • 890-SET-SIGNED-FILTER-VALUE2 • 890-SET-SIGNED-FILTER-VALUE3 • 890-SET-SIGNED-FILTER-VALUE4 • 890-SET-SIGNED-FILTER-VALUE5 • 890-SET-SIGNED-FILTER-VALUE6 • 890-SET-SIGNED-FILTER-VALUE7

Description

Sets the filter for subsequent database calls.

Using the API in a Program

There are working storage values defined for data storage depending on what you are setting. The “?” symbols in the filter are declared in order using the SET-<Type>-FILTER-VALUE commands. You move the value you want set to the working storage corresponding to that call, and then you perform that call. Set all “?” values in the filter string.

Programming Example

In the example, the database declaration (**dbdef**) has six decimal places. Because of the use of six decimal places, the SIGNED6-FILTER-VALUE in working storage is used, and then the call 890-SET-SIGNED-FILTER-VALUE6. There is a separate call for every signed value for the number of decimal places 0 to 7. The total number of digits is limited to 18 for signed values and 18 for numeric values.

```
MOVE "JKL"               TO ALPHANUM-FILTER-VALUE
PERFORM 890-SET-ALPHANUM-FILTER-VALUE
MOVE -1.2                TO SIGNED6-FILTER-VALUE
PERFORM 890-SET-SIGNED-FILTER-VALUE6
```

Using the Filter Database API Routines

This section describes the function and requirements of the filter API routines.

850-FILTER-NLT-<Index>

Name

850-FILTER-NLT-<Index>

Description

Makes the database call.

850-FILTER-NLT-<Index> reads the record whose key field is greater than or equal to the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields and the filter created earlier in the program.

Input Values

Field	Description
<Index>	You must specify the index in the statement. 850-FILTER-NLT-<Index> does not need to point to an existing record. If you define a generic value for the record search, the routine returns a record and sets the <FileName>-FOUND value for any database record successfully read.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. You do not need to fill all **DB** fields with actual values, because the routine does not need to point to an existing record. Instead, fill those fields that are not required as delimiters with zeros or spaces, depending on whether the fields are numeric or alphanumeric fields. If you need to know whether the record exactly matches the key fields you passed, you must check to see if the record matches the generic values you set.

Field	Description
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

Once the filter is defined and set, make a call to the database API using the Index Filter calls just like a Find:

```
PERFORM 850-FILTER-NLT-FLTSET1
```

850-FILTER-BEGRNG-<Index>

Name

850-FILTER-BEGRNG-<Index>

Description

Makes the database call.

Using the API in a Program

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to the **WS-DB-BEG-RNG**.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.

Field	Description
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

See the programming example for "850-FILTER-MIDRNG-<Index>" on page 159.

850-FILTER-SUBRNG-<Index>

Name

850-FILTER-SUBRNG-<Index>

Description

Makes the database call.

Using the API in a Program

The database routines perform an equal condition on all **DB** Index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a “where greater-than and less-than” condition on this index field.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.

Field	Description
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
WS-DB-SUB-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-SUB-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the index.
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

See the programming example for "850-FILTER-MIDSUB-<Index>" on page 161.

850-FILTER-MIDRNG-<Index>

Name

850-FILTER-MIDRNG-<Index>

Description

When you need to process a filtered range of data, starting at a given point in the range, use 850-FILTER-MIDRNG-<Index>.

Using the API in a Program

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to **WS-DB-BEG-RNG**.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example uses 850-FILTER-BEGRNG-<Index> and then 850-FILTER-MIDRNG-<Index> to reposition.

```
MOVE "(ARH-ORIG-AMT > ?)"      TO FILTER-STRING.
PERFORM 890-CREATE-FILTER.
MOVE 45                        TO  SIGNED2-FILTER-VALUE.
PERFORM 890-SET-SIGNED-FILTER-VALUE2.
IF (SMIDF1-FC = "I")
  MOVE SMIDF1-SV-ARH-TRANS-TYPE  TO DB-ALT-TYPE
  MOVE SMIDF1-SV-ARH-COMPANY     TO DB-COMPANY
  MOVE ARHSET6-ALT-TYPE          TO WS-DB-BEG-RNG
  PERFORM 850-FILTER-BEGRNG-ARHSET6
ELSE
IF (SMIDF1-FC = "+")
  MOVE SMIDF1-PT-ARH-TRANS-TYPE  TO DB-ALT-TYPE
  MOVE SMIDF1-PT-ARH-COMPANY     TO DB-COMPANY
  MOVE SMIDF1-PT-ARH-INVOICE     TO DB-INVOICE
  MOVE ARHSET6-INVOICE           TO WS-DB-BEG-RNG
PERFORM 850-FILTER-MIDRNG-ARHSET6
```

850-FILTER-MIDSUB-<Index>

Name

850-FILTER-MIDSUB-<Index>

Description

When you need to process a range of data, starting at a given point in the range, use 850-FILTER-MIDSUB-<Index>.

Using the API in a Program

The database routines perform an equal condition on all **DB** index fields up to the subrange field designated in the **WS-DB-SUB-RNG**. It performs a “where greater-than and less-than” condition on this index field.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

The following example uses 850-FILTER-SUBRNG-<Index> and then 850-FILTER-MIDSUB-<Index> to reposition.

```
MOVE "(ARH-ORIG-AMT > ?)"        TO FILTER-STRING.
PERFORM 890-CREATE-FILTER.
MOVE 45                          TO  SIGNED2-FILTER-VALUE.
PERFORM 890-SET-SIGNED-FILTER-VALUE2.
IF (SMIDF1-FC = "I")
  MOVE SMIDF1-SV-ARH-TRANS-TYPE  TO DB-ALT-TYPE
  MOVE SMIDF1-SV-ARH-COMPANY     TO DB-COMPANY
  MOVE WS-SUB-BEG                TO DB-INVOICE
  MOVE WS-SUB-END                TO DBRNG-INVOICE
  MOVE ARHSET6-INVOICE           TO WS-DB-SUB-RNG
  PERFORM 850-FILTER-SUBRNG-ARHSET6
ELSE
IF (SMIDF1-FC = "+")
  MOVE SMIDF1-PT-ARH-TRANS-TYPE  TO DB-ALT-TYPE
  MOVE SMIDF1-PT-ARH-COMPANY     TO DB-COMPANY
  MOVE SMIDF1-PT-ARH-INVOICE     TO DB-INVOICE
  MOVE WS-SUB-BEG                TO DBSRNG-INVOICE
  MOVE WS-SUB-END                TO DBRNG-INVOICE
  MOVE ARHSET6-INVOICE           TO WS-DB-SUB-RNG
  PERFORM 850-FILTER-MIDSUB-ARHSET6
```

850-MODFILTER-NLT-<Index>

Name

850-MODFILTER-NLT-<Index>

Description

Makes the database call and locks the record.

850-MODFILTER-NLT-<Index> reads and locks the record whose key field is greater than or equal to the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields and the filter created earlier in the program.

Input Values

Field	Description
<Index>	You must specify the index in the statement. 850-MODFILTER-NLT-<Index> does not need to point to an existing record. If you define a generic value for the record search, the routine returns a record and sets the <FileName>-FOUND value for any database record successfully read.
Record retrieval variables (also known as **DB** fields)	You must populate the index key fields used to locate the record before using this call. You do not need to fill all **DB** fields with actual values, because the routine does not need to point to an existing record. Instead, fill those fields that are not required as delimiters with zeros or spaces, depending on whether the fields are numeric or alphanumeric fields. If you need to know whether the record exactly matches the key fields you passed, you must check to see if the record matches the generic values you set.

Field	Description
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

Once the filter is defined and set, make a call to the database API using the Index Filter calls just like a Find:

```
2000-START.

    IF (ERROR-FOUND)
        GO TO 2000-SECTION-END.

    INITIALIZE                          POBMLWS-LOCAL-STORAGE.

    IF (POBMLWS-OBJ-TYPE = SPACES)
        MOVE POBMLWS-BUYER-OBJ-TYPE     TO POBMLWS-OBJ-TYPE.

    MOVE "(BUY-BUY-OBJ-ID = ?)"     TO FILTER-STRING.
    PERFORM 890-CREATE-FILTER.

    MOVE ZERO                       TO NUMERIC-FILTER-VALUE.
    PERFORM 890-SET-NUMERIC-FILTER-VALUE.

    PERFORM 910-BEGIN-SUB-TRANSACTION.
    INITIALIZE                          DB-PROCURE-GROUP
                                               DB-BUYER-CODE.
    PERFORM 850-MODFILTER-NLT-BUYSET1.
    PERFORM UNTIL (BUYER-NOTFOUND)
```

850-MODFILTER-BEGRNG-<Index>

Name

850-MODFILTER-BEGRNG-<Index>

Description

Makes the database call and locks the record.

850-MODFILTER-BEGRNG-<Index> reads and locks the first database record in the range based on the values you set in the record retrieval variables (**DB** fields) corresponding to the index key fields and the filter created earlier in the program.

Using the API in a Program

The database routines perform an equal condition on all **DB** fields that have an assigned value—that is, all columns up to, and including, the field moved to the **WS-DB-BEG-RNG**.

Input Values

Field	Description
<Index>	You must specify the index in the FIND statement.
Record retrieval variables (also known as **DB** fields)	Fill in only the **DB** fields that define the appropriate range; leave empty those **DB** fields that do not define the range.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
Filter value parameter	Set this parameter using a method described in "Setting the Filter Parameter Value" on page 150, and create the filter using the routine described in "890-CREATE-FILTER" on page 153.

Return Values

Field	Description
<FileName>-SW	The Find routines set the <FileName>-SW switch to indicate the outcome of the read.
<FileName>-NOTFOUND **<FileName>-FOUND**	Each routine sets the logical value <FileName>-NOTFOUND or <FileName>-FOUND based on the outcome of the read process.

Programming Example

This example is from PR33.

```
MOVE WS-FALSE TO  PRTRD-LP-MESSAGE-SW.

PERFORM
     VARYING PRTRD-LP-I1 FROM 1 BY 1
     UNTIL (PRTRD-LP-I1 > PRTRD-LP-DEL-MAX)
     OR    (PRTRD-LP-DEL-PROC-LEV (PRTRD-LP-I1) = SPACES)
     OR    (PRTRD-LP-MESSAGE)
          MOVE SPACES               TO DB-CYCLE-PROCESS
          MOVE SPACES               TO DB-PROC-GROUP
          MOVE PRTRD-LP-DEL-PROC-LEV (PRTRD-LP-I1)
                                     TO DB-PROCESS-LEVEL
          MOVE SPACES               TO DB-BANK-CODE
          MOVE SPACES                   TO FILTER-STRING
          MOVE "(PMN-LP140-RUN-FLG = ?)" TO FILTER-STRING
          PERFORM 890-CREATE-FILTER
          MOVE "*"                TO ALPHANUM-FILTER-VALUE
          PERFORM 890-SET-ALPHANUM-FILTER-VALUE
          MOVE PMNSET1-BANK-CODE   TO WS-DB-BEG-RNG
          PERFORM 850-MODFILTER-BEGRNG-PMNSET1
          IF (PRMONITOR-FOUND)
              MOVE "4"         TO PMN-LP140-RUN-FLG
              PERFORM 820-STORE-PRMONITOR
              MOVE WS-TRUE     TO PRTRD-LP-MESSAGE-SW
              MOVE "PRTRD"     TO CRT-ERROR-CAT
              MOVE 257         TO CRT-MSG-NBR
              PERFORM 790-GET-MSG
              MOVE CRT-MESSAGE TO PRTRD-LP-MSG
          END-IF
END-PERFORM.
```

Reusing A Filter

This section describes how to reuse a filter that you have created once within a program.

How Can I Reuse a Filter within a Program?

You can avoid recreating a filter every time you need to use it (for example, if the filter call is inside a loop). That is, you can create the filter before the loop and it remains valid until it is overwritten by a later call to 890-CREATE FILTER.

If the call to 890-CREATE FILTER is outside a loop and is thus not being recreated with each iteration through the loop, there is the limitation that the 890-SET-<Type>-FILTER-VALUE APIs do not allow you to change a value unless you also recreate the filter. To overcome this limitation, use the SETFILTERPARAM function. This function allows you to bind the parameter marker “?” to a working storage location. When you bind the parameter to working storage, the value of the working storage field is re-read every time you call one of the 850-FILTER APIs.

Like the 890-SET-<Type>-FILTER-VALUE APIs, calls to SETFILTERPARAM are matched to parameter markers (?) left to right. So if your **FILTER-STRING** is:

```
(GAM-COMPANY = ?) AND (GAM-VAR-LEVELS = ?)
```

then the first call to SETFILTERPARAM is for the GAM-COMPANY parameter marker and the second call is for the GAM-VAR-LEVELS parameter marker.

One major difference between SETFILTERPARAM and the various 890-SET-<Type>-FILTER-VALUE APIs is that SETFILTERPARAM must be called before 890-CREATE-FILTER. The 890-SET-<Type>-FILTER-VALUE APIs can be called before or after calling 890-CREATE-FILTER.

Another difference is that when you use the SETFILTERPARAM API, the format of the working storage field that you bind to must exactly match the comparison field. That is, if GAM-COMPANY is a PIC 9(4) field, the field that you bind to (GLT-TO-COMPANY in this example) must also be a PIC 9(4) field. The results of binding to a mismatched field are undefined. When you MOVE a value to one of the FILTER-VALUE fields and then call one of the 890-SET-<Type>-FILTER-VALUE APIs, the run-time system will convert the value to the appropriate type.

Update Range and Filter Update Range Routines

This chapter includes the following topics:

Setting the Update Text String

The update text string (**WS-DB-UPDATE-CMD**) is set as shown in the following example:

```
STRING
field [operation [value ...] operation [value ...] ...]
field [operation [value ...] operation [value ...] ...]
...
INTO WS-DB-UPDATE-CMD
```

where

Parameter	Description
field	Use the format *fieldname*`{[`*occurrence*`]}`, where *occurrence* is optional and specifies the number of occurrences for the field. For example: `CAPFLD[3]`
operation	Use any one of the following operators: `=`, `+=`, `-=`, `/=`, or `*=` Calculations are evaluated using standard mathematical rules, such as evaluation of expressions within parentheses first and processing multiplication before addition.
value	Use one of the following methods to set the value • "Supplying a Literal Value" on page 167 • "Specifying a Field Name" on page 168 • "Specifying a Related Field Using Aggregate Functions" on page 168

Supplying a Literal Value

To supply a literal value for the update text string, use the form *'literal'*, where the following rules apply to the *literal* value:

- A single quote (**'**) is required around the string that is a fixed value. Any single quote (**'**) embedded in the string must be escaped by an additional single quote (**' '**)
- Numbers are surrounded with single quotes (**'**) for ease and clarity of processing.
- The following operations may be used: **+**, **-**, **/**, *****
- Grouping within equations with the **and** operator is allowed.

Specifying a Field Name

To supply a field name for the update text string, use the form

*field***[***occurrence***]**

For example, **CAPFLD[3]** specifies the third occurrence of the CAPFLD field.

Specifying a Related Field Using Aggregate Functions

NOTE Fields from directly related files may be used as values, but you cannot use fields from other indirectly related files that are not directly specified in the relationship.

To supply a value from a related field, use the form

*relation***@***function***(***field***)**

where

Parameter	Description
field	Use the format *fieldname***{[***occurrence***]}**, where *occurrence* is optional and specifies the number of occurrences for the field. For example: **CAPFLD[3]** **NOTE** The square brackets enclose the number of occurrences.
function	Optional. One of the following aggregate functions: **MIN**, **MAX**, **AVG**, or **SUM**.
relation	A relationship name as defined in the primary file being updated. The file being updated must be the primary file in the relationship. **IMPORTANT** If the related field returns no values and the field to be updated can not be NULL, the update will fail.

Example: One-to-One Relationship

The following example requires the relationship RELHR be a One-to-One to the database file being updated.

```
Field = RELHR@EMPLOYEE
```

Example: One-to-Many Relationship

The following example updates from multiple records in the related file.

```
Field = RELHR@SUM(SALARY[1])
```

Update String Formatting Rules and Examples

Consider the following rules when specifying the update text string (**WS-DB-UPDATE-CMD**):

- Calculations are evaluated using standard mathematical rules, such as evaluation of expressions within parentheses first and processing multiplication before addition.
- The text string allows any of the valid types of numeric calculations (add, subtract, multiply, divide, replace) and replacements for character strings.
- Occurring fields may be updated.
- Fields specified in the update string must be database fields.
- Negative fields (for example, **-*field***) may not be used.
- Use a space between entries to ensure that text is clear. Use one space between entries, and do not use tabs or new line characters.

  For example, the text

  EV-=’1’

  could be read as

  the field **EV-** set to **1**

  – or –

  the field **EV** set to **(EV – 1)**

Examples: Update Text String Literal Values

```
TOTALPAY = HOURS * ('2' + RELNAME@SUM(SALARY))
```

```
EV1 *= '17', EV6 = EV6 * (EV3REL1@SUM(EV1) + EV10[2]), EV2 = 'RRRR'
```

NOTE The occurring field is specified as **EV10[2]**, using square brackets.

Examples: Update Text String Related Field Values

```
RELHR@SUM(SALARY[1])
```

```
RELHR@EMPLOYEE
```

Example: Setting the Text String Value

```
STRING
         "STATUS = '4', CHECK-ID = '" DELIMITED BY SIZE
         WS-U-TRD-CHECK-ID (I8)    DELIMITED BY SIZE
         "'"             DELIMITED BY SIZE
         INTO WS-DB-UPDATE-CMD
```

820-UPDATERNG-<Index>

Name

820-UPDATERNG-<Index>

Description

This API updates specified fields of the records that match the key and filter conditions, using specified actions and values. Use this API for updates that do not require the original value of a field to be read into the program. For example, to add $20 to an account, the amount originally in the account need not be known.

A return field indicates whether any rows were updated. Also, an error code is returned. Attempts to update a set of rows that cause an error (such as duplicate rows) will cause the update to fail for all of the rows.

Location

The syntax for this routine is generated when a program is compiled.

Input Values

Field	Description
<Index>	You must specify the index name that will be used to locate the records that will be updated.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
WS-DB-UPDATE-CMD	You must specify the update string that will be used in updating the records. **NOTE** The same working storage value can be used as the update value for more than one field. This is equivalent to using a MOVE or ADD function with the same value on multiple fields.

Return Values

Field	Description
WS-DB-UPDATE-COUNT	The working storage field that returns 1 or 0 depending on whether any records were updated.

Using the API in a Program

- The 820-UPDATERNG-<Index> routine can only be called from within a program transaction.
- The 820-UPDATERNG-<Index> routine locks the record(s), so there is no need to use a 840-FIND-<Index> or similar routine before performing the update.

- 820-UPDATERNG-<Index> does not have a subrange equivalent. To restrict the records beyond the range keys, you can use 820-FILTER-UPDRNG-<Index>, as described in "820-FILTER-UPDRNG-<Index>" on page 171.

Programming Example

```
PERFORM
  VARYING I8 FROM 1 BY 1
  UNTIL  (I8 > WS-U-TRD-TABLE-SIZE)
  OR     (WS-U-TRD-TIME-SEQ (I8) = ZEROES)

  STRING
     "STATUS = '4', CHECK-ID = '" DELIMITED BY SIZE
     WS-U-TRD-CHECK-ID (I8)       DELIMITED BY SIZE
     "'"                          DELIMITED BY SIZE
     INTO WS-DB-UPDATE-CMD
  MOVE WS-CURRENT-EMPLOYEE    TO DB-EMPLOYEE
  MOVE WS-U-TRD-TIME-SEQ (I8) TO DB-TIME-SEQ
  MOVE TRDSET1-TIME-SEQ       TO WS-DB-BEG-RNG

  PERFORM 820-UPDATERNG-TRDSET1
END-PERFORM
```

820-FILTER-UPDRNG-<Index>

Name

820-FILTER-UPDRNG-<Index>

Description

This API extends the functionality of the 820-UPDATERNG-<Index> API by providing filter capabilities.

You can use 820-FILTER-UPDRNG--<Index> to restrict the records beyond the range keys set for 820-UPDATERNG-<Index>. A return field indicates whether any rows were updated. Also, an error code is returned.

Location

This API is generated by the compiler.

Input Values

The same working storage value can be used as the update value for more than one field. This is equivalent to using a MOVE or ADD function with the same value on multiple fields.

Field	Description
<Index>	You must specify the database index name that you want to create a record in.
Record retrieval variables (also known as **DB** fields)	Define the beginning and ending values of the last key by moving the appropriate values to the record retrieval fields named **DB-**<KeyFieldName> and **DBRNG-**<KeyFieldName>.

Field	Description
WS-DB-BEG-RNG	Move the symbolic constant <IndexName>-<IndexFieldName> to the field WS-DB-BEG-RNG in order to tell the interface which key is the last significant one defining the range. The symbolic constant is numerically equivalent to the position of the named field in the Index.
WS-DB-UPDATE-CMD	You must specify the update string that will be used in updating the records.

Return Values

Field	Description
WS-DB-UPDATE-COUNT	The working storage field that returns 1 or 0 depending on whether any records were updated.

Using the API in a Program

- Create a filter, as described in "Database Index Filter Routines" on page 148.

Programming Example

```
.. create WS-DB-UPDATE-CMD text ...

MOVE CURRENT-COMPANY     TO DB-COMPANY.
MOVE CURRENT-EMPLOYEE     TO DB-EMPLOYEE.
MOVE TRDSET1-EMPLOYEE     TO WS-DB-BEG-RNG.

MOVE "(TRD-TIME-SEQ >= ?) AND (TRD-TIME-SEQ <= ?)"
                          TO FILTER-STRING.

... set parameter values ...
PERFORM 890-CREATE-FILTER.

PERFORM 820-FILTER-UPDRNG-TRDSET1.
```

Chapter 9

CSV File Processing Routines

Using CSV Routines

The routines described in this chapter give you access to standard methods for reading or writing Comma-Separated Value (CSV) data. This chapter also describes limitations on using the CSV routines, and gives examples of how to use them.

After you define a work file as a comma-separated variable (CSV) file using Work File Definition, you can use the following routines to open, close, read from and write to CSV files, or to change the name or type of the CSV work file.

The syntax for these routines is generated when a program is compiled. You do not need to specify any parameters for these APIs; you only need to make the API call with the correct file name. All other configuration is completed using the Workfile Definition (workdef) form as described in the *Application Development Workbench* guide.

Limitations on Using CSV Work Files

The application program does not know that this file exists, therefore the following limitations apply to CSV files manipulated by application programs.

- You can read from an existing file, append to an existing file, or write to a new file.
- You must read, write or append to a CSV work file outside of a transaction state.
- The first read or write you perform on a previously unopened CSV work file must also be outside of a transaction state.

CSV Global Variables

CSV File Information (PrefixINFO) Data Structure

The variables described in this section are used by the CSV APIs and contain information about the CSV file.

NOTE These CSV routines use the same wild card values in the external file names and the data formatting as the **lashow** CSV extraction utility.

NOTE

Field	Description
<Prefix>INFO-NBR-FIELDS	The number of fields defined in the file
<Prefix>INFO-RECORD-COUNT	The number of records in the file.
<Prefix>INFO-HANDLE	The file handle. Up to 10 characters.
<Prefix>INFO-FILE-OPEN	Indicates the status of the file. Y = the file is open. N = the file is closed. The default is N.
<Prefix>INFO-FOUND	Indicates whether the file exists.
<Prefix>INFO-FILENAME	The file name. The name can be up to 60 characters.
<Prefix>INFO-INFO-SET	One character.

Field	Description
<Prefix>INFO-FILE-OPTS	A field made up of the following subfields: • **<Prefix>INFO-HAS-HEADER** The default is Y. • **<Prefix>INFO-XLT-HEADER** The default is Y. • **<Prefix>INFO-DATE-FMT** The default is CCYYMMDD. • **<Prefix>INFO-FLD-SEP** The default is a comma. • **<Prefix>INFO-DATE-SEP** The default is a forward slash. • **<Prefix>INFO-TIME-SEP** The default is a colon. • **<Prefix>INFO-DEC-SEP** The default is a period. • **<Prefix>INFO-THOU-SEP** The default is no separator. • **<Prefix>INFO-QUOTE-CHAR** Lawson 4GL) The default is a double quote.
	• **<Prefix>INFO-FILE-TERM** The file termination character. • **<Prefix>INFO-LINE-TERM** The line termination character. • **<Prefix>INFO-MISC-OPTS**
<Prefix>INFO-OLD-FLD-NAME	The 30-character field name to be changed is moved to the OLD-FLD-NAME field. If this value is spaces, the corresponding value on the file's FLD-DESC field is not changed.
<Prefix>INFO-NEW-FLD-NAME	The new 30-character field name is moved to the NEW-FLD-NAME field. If this value is spaces, the corresponding value on the file's FLD-DESC field is not changed. The new field name is also used in matching columns and fields from an input CSV file. The matching field names and type values are not case sensitive.
<Prefix>INFO-NEW-FLD-TYPE	The new 10-character field type is entered as the text name for an element type.

Error Return Fields

If an error occurs, the following fields in the INFO data structure are populated.

Field	Description
<Prefix>INFO-ERROR-NBR (Lawson 4GL)	A 3-digit error number.
<Prefix>INFO-FIELD-NBR (Lawson 4GL)	The 3-digit number that identifies the location of a field in the file.
<Prefix>INFO-MESSAGE (Lawson 4GL)	A 60-character message.
<Prefix>INFO-ERROR (Lawson 4GL)	A 1-character field that indicates whether an error occurred.

Use Flag (UF) Fields

Each field in a file record in working storage has a corresponding a use flag, defined in a data structure. To write to a field, its associated use flag must be nonzero.

Field	Description
<Prefix>-<FieldName>-UF (Lawson 4GL) **P<Prefix><RPGFieldName>-UF** (Lawson RPG)	If two or more use flags have the same nonzero value, 800-WRITECSV-<FileName> writes the fields in file-record order. If the nonzero values are unique, 800-WRITECSV-<FileName> writes the fields using the order set by the values in the use flag fields. For more information, see "800-WRITECSV" on page 181 On input files (where Usage is defined as Input or Input/Output), the value of the use flag determines how a record is read into working storage. If the flag is set to not used, the record is initialized. For more information, see "800-READCSV" on page 182 To set all use flags at once, use "800-ALLUSEDCSV" on page 179

CSV File Access

800-OPENOUTPUTCSV

Name

800-OPENOUTPUTCSV-<FileName>

Description

This API opens a new CSV file for writing.

Programming Example

For more information, see "Programming Example: Writing to a CSV File" on page 182.

800-OPENINPUTCSV

Name

800-OPENINPUTCSV-<FileName>

Description

This API opens an existing CSV file for reading.

Programming Example

For more information, see "Programming Example: Reading Data from a CSV File" on page 183.

800-OPENAPPENDCSV

Name

800-OPENAPPENDCSV-<FileName>

Description

This API opens an existing CSV file for appending data.

800-CLOSECSV

Name

800-CLOSECSV-<FileName>

Description

This API closes the specified CSV file.

Programming Examples

For examples of closing CSV files, see

- "Programming Example: Reading Data from a CSV File" on page 183
- "Programming Example: Writing to a CSV File" on page 182

Setting CSV File Attributes

800-ALLUSEDCSV

Name

800-ALLUSEDCSV-<FileName>

Description

This API closes the specified CSV file and sets the use flag for all fields. If this routine is not called and no other modifications to the file's use flag fields are made, then all fields from the file are used by default.

Return Values

Field	Description
<Prefix>-<FieldName>-UF	Use flags set to 1 on all fields.

Programming Example

For more information, see "Programming Example: Writing to a CSV File" on page 182.

800-OVRDESCCSV

Name

800-OVRDESCCSV-<FileName>

Description

This API overrides the Field Header Name, Type, or both for one field in the specified file.

Both name and type can be changed in the same call. Once a field name has been changed, any future call to this routine for that field must use the new name. The name of the Working Storage field in the files REC area remains unchanged.

Any overrides must be done before the first write or read call to the file is made.

Input Values

Field	Description
<Prefix>INFO-OLD-FLD-NAME	The name of the field to be changed is moved to the OLD-FLD-NAME field. If this value is spaces, the corresponding value on the file's FLD-DESC field is not changed. If the Old Field Name entered does not match any fields on the file, the change is ignored and a message is written to the job log. Changing the field name of an occurring field changes all occurrences of the Old Field Name, substituting the new name but retaining the 01, 02... suffixes.
<Prefix>INFO-NEW-FLD-NAME	The new name is moved to the NEW-FLD-NAME field. If this value is spaces, the corresponding value on the file's FLD-DESC field is not changed. The new field name is also used in matching columns and fields from an input CSV file. The matching field names and type values are not case sensitive.
<Prefix>INFO-NEW-FLD-TYPE	The new field type is entered as the text name for an element type. Valid entries for NEW-FLD-TYPE are time, telephone and numeric. Alpha fields cannot be changed to numeric. Changes from numeric to time or telephone are edited for size. If an invalid type is entered or the Old and New field types are not compatible, the type is not changed and a warning message will be written to the job log. No error is returned to the program.

Return Values

The name and/or type of the CSV file is changed.

Programming Example

- "Programming Example: Reading Data from a CSV File" on page 183
- "Programming Example: Writing to a CSV File" on page 182

CSV Read/Write Processing

800-WRITECSV

Name

800-WRITECSV-<FileName>

Description

On the first call, this API opens the file for output if it is not already open. If it has a header, it formats and writes the header, and determines from the use flags the order the fields are to be written (for information, see "Use Flag (UF) Fields" on page 176).

NOTE the first write to an unopened file cannot occur within a transaction state.

On subsequent calls, it formats and writes a record with data from working storage (REC area), writing only those fields with use flags on.

Return Values

One record is written in the CSV work file.

Programming Example: Writing to a CSV File

```
    PERFORM 800-OPENOUTPUTCSV-CSVOUT1.
    PERFORM 800-ALLUSEDCSV-CSVOUT1.

    MOVE WS-TRUE               TO CVO-COMPANY-UF
                                  CVO-PROC-LEVEL-UF
                                  CVO-AMOUNT-UF
                                  CVO-DATE-UF.

    MOVE "COMPANY"     TO CVOINFO-NEW-FLD-NAME.
    MOVE "Warehouse"   TO CVOINFO-NEW-FLD-NAME.
    MOVE "numeric"     TO CVOINFO-NEW-FLD-TYPE.
    PERFORM 800-OVRDESCCSV-CSVOUT1.

    PERFORM 1000-WRITE-CSV-LOOP
    THRU    1000-LOOP-END
    UNTIL   .....

    PERFORM 800-CLOSECSV-CSVOUT1
    STOP RUN.

100-END.
      .
      .
1000-WRITE-CSV-LOOP.

    MOVE API-COMPANY           TO CVO-COMPANY.
    MOVE API-PROC-LEVEL        TO CVO-PROC-LEVEL.
    MOVE API-INVOICE-AMT       TO CVO-AMOUNT.
    MOVE API-INVOICE-DATE      TO CVO-DATE.

    PERFORM 800-WRITECSV-CSVOUT1.
      .
      .
      .
    1000-WRITE-LOOP-END.
```

800-READCSV

Name

800-READCSV-<FileName>

Description

On the first call, this API opens the file for input, if it is not already open. If there is a header, the routine reads the header and determines that fields are to be read and in what order. On subsequent calls, it reads a record into working storage (REC area), initializing any fields not flagged as used.

Return Values

One record is read from the CSV work file.

Programming Example: Reading Data from a CSV File

```
PROCEDURE DIVISION

       100-PROGRAM-CONTROL.
            .
            .
           PERFORM 800-OPENINPUTCSV-CSVOUT1.


           PERFORM 1000-READ-CSV-LOOP
           THRU    1000-LOOP-END
           UNTIL   .....

           PERFORM 800-CLOSECSV-CSVOUT1
           STOP RUN.

       100-END.
             .
             .
       1000-READ-CSV-LOOP.

           PERFORM 800-READCSV-CSVOUT1.

           MOVE CVO-COMPANY            TO API-COMPANY.
           MOVE CVO-PROC-LEVEL         TO API-PROC-LEVEL.
           MOVE CVO-INVOICE-AMT        TO API-INVOICE-AMT.
           MOVE CVO-INVOICE-DATE       TO API-INVOICE-DATE.
             .
             .
           1000-READ-LOOP-END.
```

Chapter 10

Comment and URL Attachment Processing Routines

Overview of Attachment Processing

Attachments associate additional information with a database record. This additional information takes two forms:

- Comment attachments are text
- Uniform Resource Locators (URLs) attachments are addresses to a location on the server such as a document link or an e-mail address

Attachment functionality is associated with an application form. Attachment contents are associated with the data displayed in the form and stored in the database.

The routines described in this chapter provide access to standard methods for manipulating the attachments data files. Set up of the attachment process is explained in more detail in *Application Development Workbench Standards*.

Attachment Database Files

Attachments, such as comments or URLs, are connected to data fields, records, or forms. When users add comments, the system stores the information in two attachment files that are related to the attachment-enabled data file and are named as follows:

- L_H<Prefix>
- L_D<Prefix>

where <Prefix> is the prefix of the data file to which the attachments are related; L_H indicates a header attachment file (the first 512 bytes of the attachment), and L_D indicates the detail attachment file (the remaining bytes, up to 32,000, of the attachment).

Comment Attachment Processing

Comment Attachment Variables

The following variables can be used to supply values to existing or new comments in the attachments database files.

The variable names in this table include the three-character database file prefix (<Prefix>). For example, if the program is using the file COMPANY, then the variable **CMT<Prefix>-TYPE would be CMTCOM-TYPE**.

TIP When programing a hard return, the actual field to set is called CMT<Prefix>-OPT-HARD-RTN and it has two 88 levels: CMT<Prefix>-HARD-RTN and CMT<Prefix>-NO-HARD-RTN. Set the OPT field using one of the 88 levels.

Variable	Description
CMT<Prefix>-TYPE	A one-character user-defined type of the comment as defined in the *SC*.**or** file or by the routine "900-CREATE-CMT" on page 186.
CMT<Prefix>-SIZE	The size of the comment, in bytes.
CMT<Prefix>-NAME	The name of the comment, as defined in the *SC*.**or** file or by the routine "900-CREATE-CMT" on page 186. The name can be up to 50 characters long.
CMT<Prefix>-EXT-TEXT	Up to 127 bytes of text from the comment. The amount of text put into this buffer is determined by the variable **CMT<Prefix>-EXT-TEXT-SZ.**
CMT<Prefix>-EXT-TEXT-SZ	The size of the comment text segment retrieved, stored, or appended. You can specify segments up to 127 bytes long. To preserve spacing within a comment, you can set this variable to the size of the text being appended to the comment. A value of zero in this variable tells the API to trim trailing spaces. Lawson recommends using a value of zero for this variable when the end of the comment is being appended.
CMT<Prefix>-EXT-OFFSET	The position at which to start reading text.
CMT<Prefix>-NOT-EOT	When TRUE, there is still more text in the comment that has not been read.
CMT<Prefix>-EOT	When TRUE, the end of the text in the comment has been reached.
CMT<Prefix>-OPT-HARD-RTN	When TRUE, carriage returns in the comment will be preserved.
CMT<Prefix>-OPT-SOFT-RTN	When TRUE, word wrap is applied to the text read into the **CMT<Prefix>-EXT-TEXT** buffer.
CMT<Prefix>-NO-HARD-RTN	When TRUE, carriage returns in the comment are ignored.
CMT<Prefix>-NO-SOFT-RTN	When TRUE, word wrap is not applied to the text read into the **CMT<Prefix>-EXT-TEXT** buffer.

Variable	Description
CMT<Prefix>-DATE-USER	If TRUE, it will use whatever was put into date and time fields instead of auto-filling creation and modification date and time.
CMT<Prefix>-CMT-ORDER	The sort order that determines which attachments are retrieved. Enter one of the following to sort by: 0 Name. 1 Date and time of creation. 2 Date and time of modification.
URL<Prefix>-CREATE-DATE	Date of comment creation.
URL<Prefix>-CREATE-TIME	Time of comment creation.
URL<Prefix>-CREATE-USER	User who created the comment.
URL<Prefix>-MODIFY-DATE	The last date that the comment was modified.
URL<Prefix>-MODIFY-USER	The user who last modified the comment.
URL<Prefix>-FOUND-SW	0 = Found. 1 = Not found. 3 = Error.
CMT<Prefix>-MODE	0 = Normal mode. 1 = Keyfind mode.

900-CREATE-CMT

900-CREATE-CMT-<FileName>creates a comment record for the file record currently being accessed by the program. Before creating the comment, move the user-defined comment type to the variable **CMT<Prefix>-TYPE**.

To store the comment in the database, use "900-STORE-CMT" on page 187.

Programming Example

This example sets the comment type, creates the comment, and stores the comment in the database.

NOTE The variable names in this example include the three-character database file prefix (<Prefix>). For example, if the program is using the file COMPANY, then the variable **CMT<Prefix>-TYPE would be CMTCOM-TYPE**.

```
MOVE COM-COMMENTTYPE     TO CMTCOM-TYPE.
PERFORM 900-CREATE-CMT-COMPANY.
PERFORM 900-STORE-CMT-COMPANY.
```

900-STORE-CMT

900-STORE-CMT-<FileName> commits the comment record to the database.

Using the API in a Program

Before storing a comment you should have used one of the following routines:

- "900-CREATE-CMT" on page 186
- "900-FIND-BEGRNG-CMT" on page 189
- "900-FIND-NXTRNG-CMT" on page 189

You can control the size of the comment by setting the **CMT-<Prefix>-EXT-TEXT-SZ** variable. A value of zero in this variable tells the API to trim trailing spaces.

If you will use the 900-APPEND-CMT-<FileName> API after the 900-STORE-CMT-<FileName> API, Lawson recommends that you set the size field variable.

Programming Example

This example finds a comment, retrieves the comment text, and stores the text in the database.

NOTE The variable names in this example include the three-character database file prefix (<Prefix>). For example, if the program is using the file COMPANY, then the variable **CMT<Prefix>-TYPE would be CMTCOM-TYPE**.

```
PERFORM 900-FIND-BEGRNG-CMT-COMPANY.
MOVE COM-COMMENTTEXT     TO CMTCOM-EXT-TEXT.
PERFORM 900-STORE-CMT-COMPANY.
```

900-GET-TEXT-CMT

900-GET-TEXT-CMT-<FileName> retrieves up to 127 bytes of the comment record text.

Using the API in a Program

The size of the segment retrieved can be determined by the value of the **CMT<Prefix>-EXT-TEXT-SZ** variable. You can specify segments up to 127 bytes long. To preserve spacing within a comment, you can set this variable to the size of the text being appended to the comment. A value of zero in this variable tells the API to trim trailing spaces.

You can control word wrap and hard return placement at the end of each text line retrieved using the following variables:

- CMT<Prefix>-OPT-HARD-RTN
- CMT<Prefix>-OPT-SOFT-RTN
- CMT<Prefix>-NO-HARD-RTN
- CMT<Prefix>-NO-SOFT-RTN

To check whether you have reached the end of text stored in the comment, check the value of the variable **CMT<Prefix>-NOT-EOT** or **CMT<Prefix>-EOT**.

Programming Example

This example retrieves the text of a comment.

```
MOVE 50                     TO CMTCOM-EXT-TEXT-SZ.
SET CMTCOM-HARD-RTN         TO TRUE.
SET CMTCOM-SOFT-RTN         TO TRUE.
MOVE ZEROES                 TO CMTCOM-NBR-TEXT-LINES.
SET CMTCOM-FIRST-CMT-LINE   TO TRUE.
PERFORM 900-GET-TEXT-CMT-COMPANY.
```

900-COPY-CMT

900-COPY-CMT-<FileName> copies a comment attachment from the last database file record accessed to the current database file record.

Programming Example

The example shown here copies a comment when the user chooses the U form action.

```
IF (KB01F5-FC = "U")
PERFORM 900-COPY-CMT-COMPANY.
```

900-APPEND-CMT

900-APPEND-CMT-<FileName> routine appends the text in the variable to the comment attachment for the current record.

Using the API in a Program

To preserve spacing within the comment, you can set the **CMT-<Prefix>-EXT-TEXT-SZ** variable to the size of the text being appended to the comment.

A value of zero in the **CMT-<Prefix>-EXT-TEXT-SZ** variable tells the API to trim trailing spaces. Lawson recommends using a value of zero for this variable when the end of the comment is being appended.

Programming Example

This example appends data in **CMTCOM-EXT-TEXT** to a comment attached to a record in the COMPANY file.

```
MOVE KB01F5-COM-COMMENTTEXT (I2) TO CMTCOM-EXT-TEXT
PERFORM 900-APPEND-CMT-COMPANY.
```

Comment Range Processing

900-FIND-BEGRNG-CMT

The 900-FIND-BEFRNG-CMT-<FileName> routine finds the first comment for a record.

The order of the comments (by name, date, or time) is specified by the value of **CMT<Prefix>-CMT-ORDER**.

The name stored with the comment is that of the user who created the comment.

Programming Example

The example shown here finds the comment and appends whatever is in the variable **CMT<Prefix>-EXT-TEXT**.

```
PERFORM 900-FIND-BEGRNG-CMT-COMPANY.
PERFORM 900-APPEND-CMT-COMPANY.
```

900-FIND-NXTRNG-CMT

900-FIND-NXTRNG-CMT-<FileName> finds the next comment for a record.

The order of the comments (by name, date, or time) is specified by the value of the **CMT<Prefix>-CMT-ORDER** variable.

Programming Example

The following example finds the next comment attachment, then displays the comment.

```
PERFORM 900-FIND-NXTRNG-CMT-COMPANY.
PERFORM 600-MOVE-TO-SCREEN
     THRU 600-END.
```

900-COPY-RNG-CMT

900-COPY-RNG-CMT-<FileName> copies a range of comments in the file.

Using this Routine

1. Specify the comment type to retrieve.
   - To retrieve a range of comments for a specified user-defined comment type, move a single character into **CMT<Prefix>-TYPE**.

     NOTE The character you use for user-defined types is case sensitive.

     - or -
   - To retrieve a range of all types of comments, move spaces to **CMT<Prefix>-TYPE**.

2. Locate the beginning range of the comment. Use the 900-FIND-BEGRNG-CMT-<FileName> routine.
3. Check to see if comments were found.
4. To copy the comments, use the 900-COPY-RNG-CMT-<FileName> routine.

NOTE To copy a specific comment type, set the **CMT<Prefix>-TYPE** variable to that type.

Programming Example

The following example checks the CXPTMPITEM table with a CTI prefix for all types of comments using the **CMTCTI-FOUND** variable. After checking for comments, the procedure calls 900-COPY-RNG-CMT-CXPTMPITEM.

```
MOVE SPACES                TO CTI@c1-TYPE
PERFORM 900-FIND-BEGRNG-CMT-CXPTMPITEM
IF (CTI@c1-FOUND)
     PERFORM 900-COPY-RNG-CMT-CXPTMPITEM
END-IF
```

Comment Deletion Processing

Deleting Comment Attachments

Due to the cleanup feature of the following APIs, any locks held on the parent record may be released. This occurs when a parent record no longer has an associated child attachment record. When this condition occurs, the parent record is modified to release its association with the attachments. To modify the parent record after this occurs, first re-lock the parent record.

To re-lock a record, the locking logic must follow the attachment deletion logic. If the locking logic precedes the deletion logic, an abort error occurs indicating that the previously locked record is not locked.

900-DELETE-CMT

900-DELETE-CMT-<FileName> deletes the current comment. If you need to delete a comment and then find the next comment, use "900-DELETE-RNG" on page 191.

Programming Example

The following example uses the Z form action to allow users to delete a comment from the Company file. Use the following APIs to access the comment you want to delete.

- "900-FIND-BEGRNG-CMT" on page 189
- "900-FIND-NXTRNG-CMT" on page 189

```
IF (KB01F4-FC = "Z")
PERFORM 900-DELETE-CMT-COMPANY
```

900-DELETE-RNG

900-DELETE-RNG-CMT-<FileName> deletes a range of comments from the file. Use 900-DELETE-RNG-CMT-<FileName> rather than 900-DELETE-CMT-<FileName> if you want to delete a comment and then find the next comment.

Programming Example

The following example uses the L form action to allow users to delete a range of comments.

```
PERFORM 900-FIND-BEGRNG-CMT-COMPANY
IF (KB01F4-FC = "L")
     PERFORM 900-DELETE-RNG-CMT-COMPANY
END-IF
```

URL Attachment Processing

URL Attachment Variables

The following variables can be used to supply values to existing or new URLs or e-mail address attachments in the attachments database files.

The variable names in this table include the three-character database file prefix (<Prefix>). For example, if the program is using the file COMPANY with a Prefix of COM, then the URL variable would be **URLCOM-TYPE**.

Variable	Description
URL<Prefix>-TYPE	The one-character user-defined type of the attachment as defined in the *SC*.**or** file or by the routine "900-CREATE-URL" on page 193. For more information on defining the comment type, see *Application Development Workbench Standards*.
URL<Prefix>-SIZE	The size of the attachment, in bytes.
URL<Prefix>-NAME	The name of the attachment, as defined in the *SC*.**or** file or by the routine "900-CREATE-URL" on page 193. The name can be up to 50 characters long. For more information on comment names, see *Application Development Workbench*.
URL<Prefix>-SCHEME	HTTP or e-mail address may be up to 50 characters long.
URL<Prefix>-TEXT	Text messages of 256 alpha-numeric characters.
URL<Prefix>-DATE-USER	If TRUE, it will use whatever was put into date and time fields instead of auto-filling creation and modification date and time.
URL<Prefix>-ORDER	The sort order that determines which attachments are retrieved. Enter one of the following sort: 0 Name. 1 Date and time of creation. 2 Date and time of modification.
URL<Prefix>-CREATE-DATE	Date of URL creation.
URL<Prefix>-CREATE-TIME	Time of URL creation.
URL<Prefix>-MODIFY-DATE	The last date that the URL was modified.
URL<Prefix>-FOUND-SW	0 = Found. 1 = Not found. 3 = Error.
URL<Prefix>-MODE	0 = Normal mode. 1 = Keyfind mode.

900-CREATE-URL

The 900-CREATE-URL-<FileName> routine creates a URL record for the file record currently being accessed by the program. To store the URL in the database, use "900-STORE-URL" on page 193.

Programming Example

This example creates and stores a URL for the COMPANY file.

```
PERFORM 900-CREATE-URL-COMPANY.
PERFORM 900-STORE-URL-COMPANY.
```

900-STORE-URL

900-STORE-URL-<FileName> commits the URL record to the database. Before storing a URL, you should have used one of the following routines:

- "900-CREATE-URL" on page 193
- "900-FIND-BEGRNG-CMT" on page 189
- "900-FIND-NXTRNG-URL" on page 194

Programming Example

```
PERFORM 900-FIND-BEGRNG-URL-COMPANY.
MOVE    COM-URLADDRESS          TO URLCOM-EXT-TEXT.
PERFORM 900-STORE-URL-COMPANY.
```

900-COPY-URL

900-COPY-URL-<FileName> copies a URL from the last database file record accessed to the current database file record.

Programming Example

The example shown here copies a URL when the user chooses the U form action.

```
IF (KB01F5-FC = "U")
PERFORM 900-COPY-URL-COMPANY.
```

Range URL Processing

900-FIND-BEGRNG-URL

900-FIND-BEGRNG-URL-<FileName> routine finds the first URL for a record.

The order of the URLs (by name, date, and time created or date and time modified) is specified by the value of **URL<Prefix>-ORDER**. **URL<Prefix>-TYPE** limits the find to the TYPE specified.

The name stored with the URL is that of the user who created the URL attachment.

Programming Example

The example shown here finds the first URL attachment.

```
PERFORM 900-FIND-BEGRNG-URL-COMPANY.
PERFORM 600-MOVE-TO-SCREEN
     THRU 600-END.
```

900-FIND-NXTRNG-URL

900-FIND-NXTRNG-URL-<FileName> finds the next URL for a record.

The order of the URLs (by name, date, and time created or date and time modified) is specified by the value of **URL<Prefix>-ORDER**. **URL<Prefix>-TYPE** limits the find to the type specified.

The name stored with the URL is the name given when the URL was entered, not the current user name.

Programming Example

The example shown here finds the next URL in the Company file.

```
PERFORM 900-FIND-NXTRNG-URL-COMPANY.
PERFORM 600-MOVE-TO-SCREEN
     THRU 600-END.
```

900-COPY-RNG-URL

900-COPY-RNG-URL-<FileName> copies a range of URLs in a specified file.

Using this Routine

1. To retrieve a range of URLs for a specified, user-defined type, move a single character into **URL<Prefix>-TYPE**. Note that the character you use for user-defined types is case sensitive.

   - or -

   To retrieve a range of all types of comments, move spaces to **URL<Prefix>-TYPE**.

2. To locate the beginning range of the URLs, use the 900-FIND-BEGRNG-URL-<FileName> routine.

3. Check to see if URLs were found.
4. To copy the URLs, use 900-COPY-RNG-URL-<FileName>.

To copy a specific URL type, set the **URL<Prefix>-TYPE** variable to that type.

Programming Example

The following example checks the CXPTMPITEM table with a CTI prefix. After checking for URLs, the procedure calls the 900-COPY-RNG-URL-CXPTMPITEM routine.

```
**PD
          MOVE SPACES                TO URLCTI-TYPE
          PERFORM 900-FIND-BEGRNG-URL-CXPTMPITEM
          IF (URLCTI-FOUND)
               PERFORM 900-COPY-RNG-URL-CXPTMPITEM
          END-IF
```

URL Deletion Processing

Due to the cleanup feature of the following APIs, any locks held on the parent record may be released. This occurs when a parent record no longer has an associated child attachment record. When this condition occurs, the parent record is modified to release its association with the attachments. To modify the parent record after this occurs, first re-lock the parent record.

To re-lock a record, the locking logic must follow the attachment deletion logic. If the locking logic precedes the deletion logic, an abort error occurs indicating that the previously locked record is not locked.

- "900-DELETE-URL" on page 196
- "900-DELETE-RNG-URL" on page 196

900-DELETE-URL

900-DELETE-URL-<FileName> deletes the current URL.

Programming Example

The example shown below uses a form action Z to allow the user to delete a URL from the COMPANY file. Use the following to access the URL you want to delete:

- "900-FIND-BEGRNG-URL" on page 194
- "900-FIND-NXTRNG-URL" on page 194

```
IF (KB01F4-FC = "Z")
PERFORM 900-DELETE-URL-COMPANY
```

900-DELETE-RNG-URL

900-DELETE-RNG-URL-<FileName> deletes a range of URLs in the file. Use 900-DELETE-RNG-URL-<FileName> rather than 900-DELETE-URL-<FileName> if you want to delete a comment and then find the next comment.

Programming Example

The example shown below uses a form action (L) to allow the user to delete a range of URLs.

```
PERFORM 900-FIND-BEGRNG-URL-COMPANY
IF (KB01F4-FC = "L")
     PERFORM 900-DELETE-RNG-URL-COMPANY
END-IF
```

Chapter 11

Vertex Quantum Routines

The routines described in this chapter let you make function calls from a Lawson 4GL application to determine the taxing jurisdiction and to calculate sales and use taxes. To learn more about the Vertex Quantum product, consult Vertex's Quantum documentation.

Vertex Quantum Global Variables

The variables described in this section are used by a number of Vertex APIs. Additional fields may also be used by each routine. For information on the additional fields used by a Vertex Quantum routine, see the section on that routine.

The following table describes the fields returned by Vertex Quantum routines.

VTXWS Field Descriptions

Field	Description
VTX-ERROR-STATUS	Returned by all Vertex APIs. One-character, either: N = Call successful Y = Call failed E = End of File
VTX-ERROR-MSG	Returned by all Vertex APIs. Error message of up to 60 alphanumeric characters.
VTX-GEOCODE	The GeoCode value found, up to 10 digits.
VTX-GEO-STATE	The state associated with the GeoCode, up to 25 characters.
VTX-GEO-COUNTY	The county associated with the GeoCode, up to 20 characters.
VTX-GEO-CITY	The city associated with the GeoCode, up to 25 characters.
VTX-GEO-ZIP-START	The beginning zip code associated with the GeoCode, up to 6 characters.
VTX-GEO-ZIP-END	The ending zip code associated with the GeoCode, up to 6 characters.

Field	Description
VTX-ACCESS-MODE	One alphanumeric character that indicates whether access is read-only or update. The default is R, for read-only.

Vertex Working Storage Examples

```
************************************************************
* WORKING STORAGE FOR VERTEX APIs
************************************************************
 01  VTXWS.
   02  VTX-ERROR-STATUS          PIC X(01)       VALUE ZEROES.
   02  VTX-ERROR-MSG             PIC X(60)       VALUE SPACES.
   02  VTX-GEOCODE               PIC 9(10)       VALUE ZEROES.
   02  VTX-GEO-STATE             PIC X(25)       VALUE SPACES.
   02  VTX-GEO-COUNTY            PIC X(20)       VALUE SPACES.
   02  VTX-GEO-CITY              PIC X(25)       VALUE SPACES.
   02  VTX-GEO-ZIP-START         PIC X(06)       VALUE SPACES.
   02  VTX-GEO-ZIP-END           PIC X(06)       VALUE SPACES.
   02  VTX-ACCESS-MODE           PIC X(01)       VALUE "R".

   02  VTX-NAME-CRITERIA.
     03  VTX-NAMECRTA-LEVEL    PIC  X(01)      VALUE ZEROES.
           88  NAMECRTA-LEVEL-STATE              VALUE ZEROES.
           88  NAMECRTA-LEVEL-COUNTY             VALUE "1".
           88  NAMECRTA-LEVEL-CITY               VALUE "2".
       03  VTX-NAMECRTA-STATE    PIC  X(25)      VALUE SPACES.
       03  VTX-NAMECRTA-COUNTY   PIC  X(20)      VALUE SPACES.
       03  VTX-NAMECRTA-CITY     PIC  X(25)      VALUE SPACES.
       03  VTX-NAMECRTA-ZIP      PIC  X(06)      VALUE SPACES.

   02  VTX-GEOCODE-CRITERIA.
       03  VTX-GEOCRTA-LEVEL     PIC  X(01)      VALUE SPACES.
           88  GEOCRTA-LEVEL-STATE               VALUE ZEROES.
           88  GEOCRTA-LEVEL-COUNTY              VALUE "1".
           88  GEOCRTA-LEVEL-CITY                VALUE "2".
       03  VTX-GEOCRTA-GEOCODE   PIC  9(10)      VALUE ZEROES.
       03  VTX-GEOCRTA-STATE     PIC  9(02)      VALUE ZEROES.
       03  VTX-GEOCRTA-COUNTY    PIC  9(03)      VALUE ZEROES.
       03  VTX-GEOCRTA-CITY      PIC  9(04)      VALUE ZEROES.
************************************************************
* WORKING STORAGE FOR VERTEX APIs ()
************************************************************
02  VTX-TAX-REQ.
       03  VTX-COMPANY           PIC  9(04)      VALUE ZEROES.
       03  VTX-COMPANY.
           04  VTX-TAX-CD        PIC  X(09)      VALUE SPACES.
           04  VTX-JURIS-IN-OUT  PIC  X(01)      VALUE SPACES.
      03  VTX-TAXABLE-AMT       PIC S9(13)V99
                                  SIGN IS TRAILING SEPARATE
           VALUE ZEROES.
      03  VTX-TAX-AMOUNT        PIC S9(13)V99
                                  SIGN IS TRAILING SEPARATE
           VALUE ZEROES.
      03  VTX-PROD-TAX-CAT      PIC  X(15)      VALUE SPACES.
      03  VTX-PROC-LEVEL        PIC  X(05)      VALUE SPACES.
      03  VTX-LOCATION          PIC  X(05)      VALUE SPACES.
      03  VTX-CUSTOMER-CODE.
           04  VTX-CUSTOMER      PIC  X(09)      VALUE SPACES.
           04  VTX-CUST-XTRA     PIC  X(01)      VALUE SPACES.
      03  VTX-CUST-CLASS        PIC  X(04)      VALUE SPACES.
      03  VTX-CUST-EXEMPT       PIC  X(01)      VALUE SPACES.
      03  VTX-CITY              PIC  X(18)      VALUE SPACES.
      03  VTX-STATE             PIC  X(02)      VALUE SPACES.
      03  VTX-ZIP.
           04  VTX-ZIP-FIRST-5   PIC  X(05)      VALUE SPACES.
```

```
           04  VTX-ZIP-LAST-5    PIC  X(05)      VALUE SPACES.
       03  VTX-FROM-CITY         PIC  X(18)      VALUE SPACES.
       03  VTX-FROM-STATE        PIC  X(02)      VALUE SPACES.
       03  VTX-FROM-ZIP.
           04  VTX-FR-ZIP-FIRST-5 PIC  X(05)     VALUE SPACES.
           04  VTX-FR-ZIP-LAST-5  PIC  X(05)     VALUE SPACES.
       03  VTX-FROM-TAX-CODE.
           04  VTX-FR-TAX-CD     PIC  X(09)      VALUE SPACES.
           04  VTX-FR-JURIS-IN-OUT    PIC  X(01) VALUE SPACES.
       03  VTX-INVOICE           PIC  X(22)      VALUE SPACES.
       03  VTX-LINE-NUMBER       PIC  9(06)      VALUE ZEROES.
       03  VTX-QUANTITY          PIC S9(11)V9(4)
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
       03  VTX-PROD-EXEMPT       PIC  X(01)      VALUE SPACES.
       03  VTX-THIRD-PARTY-FUNC  PIC  X(01)      VALUE SPACES.
       03  VTX-TRANS-TYPE        PIC  X(01)      VALUE SPACES.
       03  VTX-TRANS-SUB-TYPE    PIC  X(03)      VALUE SPACES.
       03  VTX-SAVE-EFFECT-DATE  PIC  9(08)      VALUE ZEROES.
       03  VTX-SYSTEM-DATE       PIC  9(08)      VALUE ZEROES.
       03  VTX-VERTEX-FLAG       PIC X(01)       VALUE SPACES.
*************************************************************
* WORKING STORAGE FOR VERTEX APIs
*************************************************************
02  VTX-TAX-RESULT.
       03  VTX-INV-TOTAL-TAX        PIC S9(10)V999
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
       03  VTX-TRANS-STATUS-CD      PIC 9(01)    VALUE ZEROES.
       03  VTX-TO-JURIS-RETURN-CD   PIC 9(01)    VALUE ZEROES.
       03  VTX-FROM-JURIS-RETURN-CD PIC 9(01)    VALUE ZEROES.

       03  VTX-STATE-LOCAL       OCCURS 4 TIMES.
           04  VTX-STLOC-TAXBLTY-FLAG        PIC X(01).
           04  VTX-STLOC-TAX-TYPE            PIC X(01).
           04  VTX-STLOC-TAX-INCLUDED-FLAG   PIC X(01).
           04  VTX-STLOC-EXEMPT-REASON-CD    PIC X(01).
           04  VTX-STLOC-EXEMPT-AMOUNT       PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-STLOC-NON-TAX-REASON-CD   PIC X(01).
           04  VTX-STLOC-NON-TAX-AMOUNT      PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-STLOC-RATE                PIC SV9(7)
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-STLOC-RATE-DATE           PIC 9(08).
           04  VTX-STLOC-TAX-AMOUNT          PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-STLOC-TAX                 PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-STLOC-DISTRICT-FLAG       PIC X(01).
       03  VTX-LOCAL-ADDITION    OCCURS 3 TIMES.
           04  VTX-LOCADD-TAXBLTY-FLAG       PIC X(02).
           04  VTX-LOCADD-TAX-TYPE           PIC X(02).
           04  VTX-LOCADD-TAX-INCLUDED-FLAG  PIC X(01).
           04  VTX-LOCADD-EXEMPT-REASON-CD   PIC X(01).
           04  VTX-LOCADD-EXEMPT-AMOUNT      PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-LOCADD-NON-TAX-REASON-CD  PIC X(01).
           04  VTX-LOCADD-NON-TAX-AMOUNT     PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-LOCADD-RATE               PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-LOCADD-RATE-DATE          PIC 9(08).
           04  VTX-LOCADD-TAX-AMOUNT         PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-LOCADD-TAX                PIC S9(13)V99
                     SIGN IS TRAILING SEPARATE VALUE ZEROES.
           04  VTX-LOCADD-TAXED-GEO-FLAG     PIC X(01).
           04  VTX-LOCADD-DISTRICT-FLAG      PIC X(01).
```

GeoCoder Routines

Use the following GeoCoder routines to determine the jurisdictions for further tax calculations.

900-GEO-CONNECT

900-GEO-CONNECT establishes either a read-only or an update connection to the Vertex GeoCoder Database. Typically, you make a read-only connection. After the connection is set up, you may call other Vertex GeoCoder routines.

Input Values

To use the 900-GEO-CONNECT routine, you must populate the following field.

Field	Description
VTX-ACCESS-MODE	One alphanumeric character that indicates whether access is read-only or update. The default is R, for read-only.

Return Values

900-GEO-CONNECT returns values in the **VTX-ERROR-STATUS** and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using the Routine

1. Populate the **VTX-ACCESS-MODE** field.
2. Call the 900-GEO-CONNECT routine.
3. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
4. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
MOVE "R" TO VTX-ACCESS-MODE.

PERFORM 900-GEO-CONNECT.
IF VTX-ERROR-STATUS = "Y"
    DISPLAY VTX-ERROR-MSG.
```

900-GEO-SET-NAME-CRITERIA

900-GEO-SET-NAME-CRITERIA sets the state, county, and city name search criteria for subsequent calls to 900-GEO-GET-NEXT-GEOCODE. The values are the actual names of the state, county, and city. Subsequent calls to the 900-GEO-GET-NEXT-GEOCODE routine return the first GeoCode in the search.

The search criteria are valid until the next call to the 900-GEO-SET-NAME-CRITERIA routine or the 900-GEO-SET-GEOCODE-CRITERIA routine.

Input Fields

Populate the following field.

Field	Description
VTX-NAMECRTA-LEVEL	Required. A one-character code that indicates the level at which to extend the search. 0 = State level searching 1 = County level searching 2 = City level searching
VTX-NAMECRTA-STATE	If you entered a value of 0 (zero) for **VTX-NAMECRTA-LEVEL**, populate this field with up to 25 alphanumeric characters for the state name.
VTX-NAMECRTA-COUNTY	If you entered a value of 1 (one) for **VTX-NAMECRTA-LEVEL**, populate this field with up to 20 alphanumeric characters for the county name.
VTX-NAMECRTA-CITY	If you entered a value of 2 (two) for **VTX-NAMECRTA-LEVEL**, populate this field with up to 25 alphanumeric characters for the city name.
VTX-NAMECRTA-ZIP	Optional. Populate this field with up to 6 numeric characters for the zip code.

Return Values

900-GEO-SET-NAME-CRITERIA returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using this Routine

1. Populate the required field and optional fields.
2. Call 900-GEO-SET-NAME-CRITERIA.
3. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
4. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
MOVE VGEOF1-VTX-STATE              TO VTX-NAMECRTA-STATE.
MOVE VGEOF1-VTX-COUNTY             TO VTX-NAMECRTA-COUNTY.
MOVE VGEOF1-VTX-CITY               TO VTX-NAMECRTA-CITY.
MOVE VGEOF1-VTX-LEVEL              TO VTX-NAMECRTA-LEVEL.

PERFORM 900-GEO-SET-NAME-CRITERIA.
IF (VTX-ERROR-STATUS = "Y")
   MOVE VTX-ERROR-MSG       TO VGEOF1-VTX-ERROR-MSG
ELSE
   PERFORM 900-GEO-GET-NEXT-GEOCODE
   IF (VTX-ERROR-STATUS = "Y")
      MOVE VTX-ERROR-MSG       TO VGEOF1-VTX-ERROR-MSG
   ELSE
      IF (VTX-ERROR-STATUS = "E")
         MOVE VTX-ERROR-MSG       TO VGEOF1-VTX-ERROR-MSG
      ELSE
         MOVE "No Error"          TO VGEOF1-VTX-ERROR-MSG.
```

900-GEO-SET-GEOCODE-CRITERIA

900-GEO-SET-GEOCODE-CRITERIA sets the state, county and city name search criteria for subsequent calls to 900-GEO-GET-NEXT-GEOCODE. The digits are assigned to codes as shown in the following table.

Position	Meaning
1-2	State
3-5	County
6-9	City
10	Inside or outside city limits

Subsequent calls to the 900-GEO-GET-NEXT-GEOCODE routine return the first GeoCode in the search.

The search criteria are valid until the next call to the 900-GEO-SET-NAME-CRITERIA routine or the 900-GEO-SET-GEOCODE-CRITERIA routine.

Input Fields

Populate the following fields.

Field	Description
VTX-GEOCRTA-LEVEL	Required. A one-character code that indicates the level at which to extend the search. 0 = State level searching 1 = County level searching 2 = City level searching
VTX-GEOCRTA-STATE	If you entered a value of 0 (zero) for **VTX-GEOCRTA-LEVEL**, populate this field with up to 2 numeric characters for the Vertex Geocode state value.

Field	Description
VTX-GEOCRTA-COUNTY	If you entered a value of 1 (one) for **VTX-GEOCRTA-LEVEL**, populate this field with up to 3 numeric characters for the Vertex Geocode county value.
VTX-GEOCRTA-CITY	If you entered a value of 2 (two) for **VTX-GEOCRTA-LEVEL**, populate this field with up to 4 numeric characters for the Vertex Geocode city value.

Return Values

The 900-GEO-SET-GEOCODE-CRITERIA routine returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using the Routine

1. Populate the required fields.
2. Call the 900-GEO-SET-GEOCODE-CRITERIA routine.
3. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
4. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
MOVE VGEOF1-VTX-GEOCODE-CODE       TO VTX-GEOCRTA-GEOCODE.
MOVE VGEOF1-VTX-STATE-CODE         TO VTX-GEOCRTA-STATE.
MOVE VGEOF1-VTX-COUNTY-CODE        TO VTX-GEOCRTA-COUNTY.
MOVE VGEOF1-VTX-CITY-CODE          TO VTX-GEOCRTA-CITY.
MOVE VGEOF1-VTX-LEVEL-CODE         TO VTX-GEOCRTA-LEVEL.

PERFORM 900-GEO-SET-GEOCODE-CRITERIA.
IF (VTX-ERROR-STATUS = "Y")
   MOVE VTX-ERROR-MSG       TO VGEOF1-VTX-ERROR-MSG
```

900-GEO-GET-NEXT-GEOCODE

900-GEO-GET-NEXT-GEOCODE returns the next GeoCode based on the criteria you set in the 900-GEO-SET-NAME-CRITERIA routine or the 900-GEO-SET-GEOCODE-CRITERIA routine.

A table is maintained that contains the list of GeoCodes that match the search criteria. The position in the list is set to point to the next GeoCode after each call. On the first call to 900-GEO-GET-NEXT-GEOCODE, the first GeoCode is returned. When no more GeoCodes are found for the given criteria, the **VTX-ERROR-STATUS** field contains E to indicate an **End of file**.

Return Values

900-GEO-GET-NEXT-GEOCODE returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

The 900-GEO-GET-NEXT-GEOCODE routine also returns values in the following fields.

Field	Description
VTX-GEOCODE	The GeoCode value found, up to 10 digits.
VTX-GEO-STATE	State name associated with the GeoCode, up to 25 alphanumeric characters.
VTX-GEO-COUNTY	County name associated with the GeoCode, up to 20 alphanumeric characters.
VTX-GEO-CITY	City name associated with the GeoCode, up to 25 alphanumeric characters.
VTX-GEO-ZIP-START	Beginning Postal code in the GeoCode, up to 6 alphanumeric characters.
VTX-GEO-ZIP-END	Ending Postal code in the GeoCode, up to 25 alphanumeric characters.

Using the Routine

The 900-GEO-GET-NEXT-GEOCODE routine is typically called within a loop which is terminated when **VTX-ERROR-STATUS** contains the value E.

1. Call the 900-GEO-SET-NAME-CRITERIA routine or the 900-GEO-SET-GEOCODE-CRITERIA routine.

   This step resets the pointer in the list so that the following call to 900-GEO-GET-NEXT-GEOCODE returns the first GeoCode. The reset occurs even if the set criteria have not changed.

2. Call the 900-GEO-GET-NEXT-GEOCODE routine.
3. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.

   If an error occurred, display the **VTX-ERROR-MSG** field.

   – or –

   If **VTX-ERROR-STATUS** = E, then no more GeoCodes exist for the set criteria.

Programming Example

For an example of 900-GEO-GET-NEXT-GEOCODE routine use in a program, see "Programming Example" on page 202

900-GEO-DISCONNECT

900-GEO-DISCONNECT removes the connection to the Vertex GeoCoder Database and closes the database files. Any subsequent calls to the Vertex GeoCoder Database will result in errors.

Return Values

900-GEO-DISCONNECT returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using the Routine

1. Call 900-GEO-DISCONNECTs.
2. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
3. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
PERFORM 900-GEO-DISCONNECT.
IF VTX-ERROR-STATUS = "Y"
    DISPLAY VTX-ERROR-MSG.
```

Sales and Use Tax Routines

The routines in this section calculate sales and use taxes.

900-OPEN-VERTEX

900-OPEN-VERTEX opens the Vertex Quantum Database files and prepares them for use. This routine must be called before either 900-CLOSE-VERTEX or 900-CALC-VERTEX-TAX is called.

Return Values

900-OPEN-VERTEX returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using the Routine

1. Call 900-OPEN-VERTEX.
2. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
3. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
PERFORM 900-OPEN-VERTEX.
IF VTX-ERROR-STATUS = "Y"
    DISPLAY VTX-ERROR-MSG.
```

900-CALC-VERTEX-TAX

900-CALC-VERTEX-TAX calculates tax information based on information passed to the routine in the **VTX-TAX-REQ** field. The routine returns results in the **VTX-TAX-RESULT** field.

Required Fields

Populate these fields before calling the routine.

Field	Definition
VTX-COMPANY	The four-character, numeric company code.

Field	Definition
VTX-TAX-CODE	Consists of the following two fields: • **VTX-TAX-CD** The 9-character numeric GeoCode for the ship-to location. • **VTX-JURIS-IN-OUT** A one character code that indicates whether ship-to location is inside (1=TRUE) or outside (0=FALSE) the city limits.
VTX-TAXABLE-AMT	The signed, 12-character (with 2 decimal places) numeric value for the total dollar amount for the line item.
VTX-TAX-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the total tax amount for the transaction. This field is used when processing special (tax-only debit and credit) transactions.
VTX-PROD-TAX-CAT	A 15-character alphanumeric product code.
VTX-PROC-LEVEL	A 5-character alphanumeric division code.
VTX-LOCATION	A 5-character alphanumeric store location code.
VTX-CUSTOMER-CODE	A 10-character alphanumeric customer code. This field is not used.
VTX-CUST-CLASS	A 4-character customer class code.
VTX-CUST-EXEMPT	A one-character flag that shows whether the customer is taxable or exempt. E = Exempt T = Taxable D = TDM Database
VTX-CITY	The 18-character alphanumeric city for the ship-to location.
VTX-STATE	The 2-character alphanumeric state abbreviation for the ship-to location.
VTX-ZIP	The 10-character alphanumeric postal code for the ship-to location.
VTX-FROM-CITY	The 18-character alphanumeric city for the ship-from location.
VTX-FROM-STATE	The 2-character alphanumeric state for the ship-from location
VTX-FROM-ZIP	The 10-character alphanumeric postal code for the ship-from location.
VTX-FROM-TAX-CODE	Consists of the following two fields: • **VTX-FR-TAX-CD** The 9-character numeric GeoCode for the ship-from location. • **VTX-FR-JURIS-IN-OUT** A one-character alphanumeric code that indicates whether ship-from location is inside (1=TRUE) or outside (0=FALSE) the city limits.

Field	Definition
VTX-INVOICE	A 22-character alphanumeric invoice number for the transaction
VTX-LINE-NUMBER	A 6-character numeric line item number for the transaction.
VTX-QUANTITY	A signed, 10-character numeric value (including 4 decimal spaces) that indicates the quantity of items for the transaction.
VTX-PROD-EXEMPT	A one-character alphanumeric code that indicates whether the product is taxable or nontaxable. You can override the **VTX-CUST-EXEMPT** flag for a specified line item. E = Exempt T = Taxable D = TDM Database
VTX-THIRD-PARTY-FUNC	A one-character alphanumeric code that indicates how the register should be updated. R = Update register O = Update register and use **VTX-TAX-AMOUNT** field
VTX-TRANS-TYPE	A one-character alphanumeric code that indicates the type of transaction. S = Sale P = Purchase
VTX-TRANS-SUB-TYPE	A 3-character alphanumeric code that further identifies the type of transaction. FRT = Freight SPACES = None
VTX-SAVE-EFFECT-DATE	An 8-character numeric invoice date in CCYYMMDD format, used to retrieve current or previous tax rates. This field is useful when processing adjustments or credits for prior invoices.
VTX-SYSTEM-DATE	An 8-character numeric date the transaction is processed, in CCYYMMDD format.
VTX-VERTEX-FLAG	A one-character alphanumeric access code that indicates whether this function is specifying the jurisdiction by GeoCode or by some combination of address information. G = GeoCode S = State, postal code C = State, postal code, city Default = State, postal code, both

Return Values

The 900-CALC-VERTEX-TAX returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

The 900-CALC-VERTEX-TAX routine also returns values in the following fields in the **VTX-TAX-RESULT** structure.

Field	Description
	The signed, 10-character numeric value (including 3 decimal spaces) that indicates the total tax, sum of return values from **VTX-STATE-LOCAL**.
VTX-TRANS-STATUS-CD	A one-character numeric code that determines hether taxes were collected for a jurisdiction other than the taxing jurisdiction.
VTX-TO-JURIS-RETURN-CD	A one-character numeric code that returns the level of the GeoCode (state, city, or none) for the ship-to location.
VTX-FROM-JURIS-RETURN-CD	A one-character numeric code that returns the level of the GeoCode (state, city, or none) for the ship-from location.
VTX-STATE-LOCAL	Occurs 4 times (state, county, city, district). For more information, see " State and Local Tax Variable Structure" on page 209.
VTX-LOCAL-ADDITION	Occurs 3 times (county, city, district). For more information, see "Local Jurisdiction Tax Structure" on page 210.

State and Local Tax Variable Structure

The following table shows the fields returned in the **VTX-STATE-LOCAL** structure.

Field	Description
VTX-STLOC-TAXBLTY-FLAG	The one-character, alphanumeric code that determines whether there is tax liability in the jurisdiction. Currently returns only 0, tax calculation process determines tax ability.
VTX-STLOC-TAX-TYPE	The one-character, alphanumeric code that indicates the type of tax collected for jurisdiction. Currently returns only 1, tax calculation process determines tax type.
VTX-STLOC-TAX-INCLUDED-FLAG	The one-character, alphanumeric code that indicates whether the tax amount is included in the extended amount (used for Canadian GST transactions). Currently returns only 0, tax amount not included in extended amount.
VTX-STLOC-EXEMPT-REASON-CD	The one-character, alphanumeric code identifying the type of exemption. Currently returns only 0.
VTX-STLOC-EXEMPT-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the dollar amount exempt from state and local taxes.
VTX-STLOC-NON-TAX-REASON-CD	The one-character, alphanumeric code that identifies the type of nontaxable transaction. Currently returns only 0.
VTX-STLOC-NON-TAX-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the nontaxable dollar amount.
VTX-STLOC-RATE	The signed, 7-character numeric value for the jurisdiction's tax rate or a user-supplied tax rate.

Field	Description
VTX-STLOC-RATE-DATE	The signed, 8-character numeric value for the effective date of the jurisdiction tax rate in CCYYMMDD format.
VTX-STLOC-TAX-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the amount subject to tax.
VTX-STLOC-TAX	The signed, 12-character (with 2 decimal places) numeric value for the tax amount.
VTX-STLOC-DISTRICT-FLAG	The one-character, alphanumeric code that determines whether the district tax, if any, is for the city or the county. Currently returns only 0, no city or county level district tax.

Local Jurisdiction Tax Structure

The following table shows the fields returned in the **VTX-LOCAL-ADDITION** structure.

Field	Definition
VTX-LOCADD-TAXBLTY-FLAG	The two-character, alphanumeric code that indicates whether there is tax liability in the jurisdiction. 00 = Tax calculation process determines tax ability 01 = Taxable 02 = Nontaxable 03 = Exempt 04 = Collect Tax In window-defined taxable situation 05 = Collect Tax In window-defined nontaxable situation 06 = Jurisdiction Exceptions window-defined taxable situation 07 = Jurisdiction Exceptions window-defined nontaxable situation 08 = GeoCode overrides used 09 = Product Exceptions window-defined taxable situation 10 = Product Exceptions window-defined nontaxable situation 11 = Product Exceptions window overrides used 12 = Customer Exceptions window-defined taxable situation 13 = Customer Exceptions window-defined nontaxable situation 14 = Customer Exceptions window overrides used 15 = Locations window-defined taxable situation 16 = Locations window-defined nontaxable situation 17 = Tax ability undefined

Field	Definition
VTX-LOCADD-TAX-TYPE	The two-character, alphanumeric code that identifies type of tax for jurisdiction. 00 = No tax type 01 = Tax calculation process should determine tax type 02 = Sales tax type 03 = Use tax type 04 = Rental or lease tax type 05 = Service tax type 06 = Override sales tax type 07 = Override use tax type 08 = Override rental or lease tax type 09 = Override service tax type 10 = Exempt sales transaction 11 = Exempt use transaction 12 = Exempt rental or lease transaction 13 = Exempt service transaction 14 = Nontaxable sales transaction 15 = Nontaxable use transaction 16 = Nontaxable rental or lease transaction 17 = Nontaxable service transaction 18 = Jurisdiction imposes no sales tax 19 = Jurisdiction imposes no use tax 20 = Jurisdiction imposes no rental or lease tax 21 = Jurisdiction imposes no service tax 22 = Invalid jurisdiction
VTX-LOCADD-TAX-INCLUDED-FLAG	The one-character, alphanumeric code that indicates whether tax amount is included in extended amount. (Used for Canadian GST transactions.) 0 = Tax amount not included in extended amount 1 = Tax amount included in extended amount
VTX-LOCADD-EXEMPT-REASON-CD	The one-character, alphanumeric code that identifies exemption type * (asterisk) = Transaction is exempt 9 = Transaction is exempt because the jurisdiction is an APO/FPO 0 = Nonexempt

Field	Definition
VTX-LOCADD-EXEMPT-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the dollar amount exempt from taxes
VTX-LOCADD-NON-TAX-REASON-CD	The one-character, alphanumeric code that identifies the type of nontaxable transaction * (asterisk) = Transaction is nontaxable and a nontaxable reason code was not supplied 0 = Transaction is taxable or exempt
VTX-LOCADD-NON-TAX-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the nontaxable amount.
VTX-LOCADD-RATE	The signed, 12-character (with 2 decimal places) numeric value for the jurisdiction tax rate.
VTX-LOCADD-RATE-DATE	The signed, 8-character numeric value for the effective date of the jurisdiction tax rate in CCYYMMDD format
VTX-LOCADD-TAX-AMOUNT	The signed, 12-character (with 2 decimal places) numeric value for the amount subject to tax.
VTX-LOCADD-TAX	The signed, 12-character (with 2 decimal places) numeric value for the tax amount.
VTX-LOCADD-TAXED-GEO-FLAG	The one-character, alphanumeric code that identifies the taxing jurisdiction 0 = No taxing jurisdiction exists 1 = Tax calculation process should determine taxing jurisdiction 2 = Ship-to location 3 = Ship-from location 4 = Order-acceptance location
VTX-LOCADD-DISTRICT-FLAG	The one-character, alphanumeric code that indicates whether any district tax is collected and at what jurisdiction level 0 = No city- or county-level district task 1 = City-level district tax applies 2 = County level district tax applies

Using the Routine

1. Populate the fields in the **VTX-TAX-REQ** structure.
2. Call the 900-CALC-VERTEX-TAX routine.
3. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
4. If an error occurred, display the **VTX-ERROR-MSG** field.
5. Return results are in the **VTX-TAX-RESULT** field.

Programming Example

```
MOVE VRTXF1-VTX-FROM-GEOCODE       TO VTX-FR-TAX-CD.
MOVE VRTXF1-VTX-TO-GEOCODE         TO VTX-TAX-CD.
MOVE VRTXF1-VTX-LINE-NBR           TO VTX-LINE-NUMBER.
MOVE VRTXF1-VTX-TAXABLE-AMOUNT     TO VTX-TAXABLE-AMT.
MOVE "G"                           TO VTX-VERTEX-FLAG.
MOVE "D"                           TO VTX-CUST-EXEMPT.
MOVE "D"                           TO VTX-PROD-EXEMPT.
MOVE "S"                           TO VTX-TRANS-TYPE.
MOVE SPACES                        TO VTX-TRANS-SUB-TYPE.
MOVE "1"                           TO VTX-JURIS-IN-OUT.
MOVE "1"                           TO VTX-FR-JURIS-IN-OUT.
PERFORM 900-CALC-VERTEX-TAX.
IF (VTX-ERROR-STATUS = "Y")
   MOVE VTX-ERROR-MSG       TO VRTXF1-VTX-ERROR-MSG
ELSE
   MOVE "No Error"          TO VRTXF1-VTX-ERROR-MSG.
```

900-CLOSE-VERTEX

900-CLOSE-VERTEX closes all Vertex Quantum Database files. Call this routine when you no longer need access to Vertex Quantum Database routines.

Return Values

900-CLOSE-VERTEX returns values in the **VTX-ERROR-STATUS**and **VTX-ERROR-MSG** fields. For information on these fields see "Vertex Quantum Global Variables" on page 197

Using the Routine

1. Call 900-CLOSE-VERTEX.
2. Examine the **VTX-ERROR-STATUS** field to determine if an error occurred.
3. If an error occurred, display the **VTX-ERROR-MSG** field.

Programming Example

```
PERFORM 900-CLOSEVERTEX.
IF VTX-ERROR-STATUS = "Y"
    DISPLAY VTX-ERROR-MSG.
```

Appendix A

Deprecated APIs

This appendix describes APIs or routines that you might see in the source code, but you should no longer use.

Deprecated Date and Time Processing

The 900 series of date and time routines are preferred over the following routines that are placed here for reference purposes.

500-DATE-COMPARE

IMPORTANT This routine is provided for backward compatibility. Use "900-NBR-DAYS-IN-DATE-RNG" on page 69 instead.

The 500-DATE-COMPARE routine converts its two input dates to Julian format and calculates their difference.

Library DATERTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	The signed, 8-character numeric value for the first date.
DATE-2	N 8	The signed, 8-character numeric value for the second date.

NOTE If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
DAYS-DIFFERENCE	N 6	Set to number of days difference between **DATE-1** and **DATE-2**; the result is positive if **DATE-1** is less than **DATE-2**.

Example

```
*       Check for date differences.
        MOVE first-date                 TO DATE-1.
        MOVE second-date                TO DATE-2.
        PERFORM 500-DATE-COMPARE.
```

501-CHECK-LEAP-YEAR

The 501-CHECK-LEAP-YEAR routine counts one for each leap year that falls between the two input dates.

NOTE Years that end in 00 are not counted as leap years, unless the year ending in 00 is divisible by 400.

Library DATERTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	First date.*
DATE-2	N 8	Second date.*

NOTE If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
DAYS-DIFFERENCE	N 6	Number of leap years between **DATE-1** and **DATE-2**.

Example

```
*       Count leap years between start and stop dates.
        MOVE start-date                 TO DATE-1.
        MOVE stop-date                  TO DATE-2.
        MOVE ZERO                       TO DAYS-DIFFERENCE.
        PERFORM 501-CHECK-LEAP-YEAR.
```

510-DATE-EDIT

IMPORTANT This routine is provided for backward compatibility. Use "900-IS-DATE-INVALID" on page 68 instead.

The 510-DATE-EDIT routine edits the input date for a valid month, day, and year. If in error, the **DATE-ERROR** switch is set to true and **CRT-ERROR-NBR** is set to indicate which error occurred. See the ""Century Parameter Processing" on page 56"

This routine replaces a century of 00 by storing the century in the input date.

Library DATERTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	Date to be changed.*

NOTE *If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56"

Output

Field	Type and length	Definition
DATE-ERROR	N 1	Set to true on error.
CRT-ERROR-NBR	N 3	If error, set to error number.
NO-ERROR-FOUND	N 9	Value zero.
ERROR-FOUND	A 20	Value 1 through 999.

Example

```
*       Edit data for valid month and day.
        MOVE input-date                 TO DATE-1.
        PERFORM 510-DATE-EDIT.
```

515-UPDATE-DATE-TIME

The 515-UPDATE-DATE-TIME routine sets **WS-DATE-TIME** to the current date and time. See the "Century Parameter Processing" on page 56

Library DATERTNS

Required Input

No required input

Output

Field	Type and length	Definition
WS-SYSTEM-TIME		Time and date structure is returned in the following formats:
WS-DATE-TIME	N 6	MMDDYY
WS-SYSTEM-DATE-YMD	N 8	YYYYMMDD
WS-SYSTEM-TIME	N 8	The time is returned as HHMMSSSS with low-order zero fill for tenths and hundredths of seconds.

Example

```
*       Update the date and clock time.
        PERFORM 515-UPDATE-DATE-TIME.
```

520-COMPUTE-JULIAN

IMPORTANT This routine is provided for backward compatibility. Use "900-DATE-TO-JULIAN" on page 58 instead.

The 520-COMPUTE-JULIAN routine calculates a Julian date number from the base date of Oct. 15, 1582.

Library JULIANRTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	Date to be changed.*

NOTE *If the input date century equals 00, it changes to 19 or 20. See "Century Parameter Processing" on page 56

Output

Field	Type and length	Definition
JULIAN-DAYS	N 6	Julian date number. Always set. May be zero.

Example

```
*       Compute a base Julian date.
        MOVE new-date                   TO DATE-1.
        PERFORM 520-COMPUTE-JULIAN.
```

520-DATE-LIT

IMPORTANT This routine is provided for backward compatibility. Use instead.

The 520-DATE-LIT routine converts the date in **DATE-1** to a display format in which the month name is substituted for the number of the month. The input month is validated before conversion.

Library DATERTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	Date to be converted.

Output

Field	Type and length	Definition
LITERAL-DATE	A 20	Converted date. Spaces = month in error.

Example

```
*       Set date for heading print.
        MOVE report-date                    TO DATE-1.
        PERFORM 520-DATE-LIT.
```

530-COMPUTE-GREGORIAN

IMPORTANT This routine is provided for backward compatibility. Use instead.

The 530-COMPUTE-GREGORIAN routine converts a Julian date number, relative to year zero, to a Gregorian date. The routine does not validate the Julian number and a negative value produces erroneous results.

Library JULIANRTNS

Required Input

Field	Type and length	Definition
JULIAN-DAYS	N 6	Julian date number.

Output

Field	Type and length	Definition
DATE-1	N 8	Gregorian date in yyyymmdd format.

Example

```
*       Convert record date back to print format.
        MOVE Julian-hold-date           TO JULIAN-DAYS.
        PERFORM 530-COMPUTE-GREGORIAN.
```

540-COMPUTE-WEEKDAY

IMPORTANT This routine is provided for backward compatibility. Use or instead, depending on the output you want.

The 540-COMPUTE-WEEKDAY routine calculates which day of the week the input date is. The input date is assumed to be valid.

Library DATERTNS

Required Input

Field	Type and length	Definition
DATE-1	N 8	Date to be computed.

Output

Field	Type and length	Definition
DAY-NBR	N 4	1 = Sunday, 2 = Monday, and so on.
WEEK-DAY	A 9	Sunday, Monday, and so on.

Example

```
*       Find day of week.
        MOVE input-date                         TO DATE-1.
        PERFORM 540-COMPUTE-WEEKDAY.
```

580-STRING-DATE-MMDD

The 580-STRING-DATE-MMDD routine strings the given date into the output area as MM/DD. No error indication is given for an invalid date.

Library STRRTNS

Required Input

Field	Type and length	Definition
WS-STRFMT-DATE	N 8	Given date.

Optional Output

Field	Type and length	Definition
WS-STRFLD-OUT	A 301	MM/DD format
WS-STRFLD-PNTR	N 4	Optional

Example

```
*       Edit date as MM/DD.
        MOVE input-date            TO WS-STRFLD-DATE.
        PERFORM 580-STRING-DATE-MMDD.
```

580-STRING-DATE-MMDDYY

The 580-STRING-DATE-MMDDYY routine strings the given date into the output area as MM/DD/YY. No error indication is given for an invalid date.

Library STRRTNS

Required Input

Field	Type and length	Definition
WS-STRFMT-DATE	N 8	Given date.

Optional Output

Field	Type and length	Definition
WS-STRFLD-OUT	A 301	MM/DD/YY format.
WS-STRFLD-PNTR	N 4	Optional.

Example

```
*       Edit date as MM/DD/YY.
        MOVE input-date           TO WS-STRFLD-DATE.
        PERFORM 580-STRING-DATE-MMDDYY.
```

580-STRING-DATE-MMDDYYYY

The 580-STRING-DATE-MMDDYYYY routine strings the given date into the output area as MM/DD/YYYY. No error indication is given for an invalid date.

Library STRRTNS

Required Input

Field	Type and length	Definition
WS-STRFMT-DATE	N 8	Given date.

Optional Output

Field	Type and length	Definition
WS-STRFLD-OUT	A 301	MM/DD/YYYY format.
WS-STRFLD-PNTR	N 4	Optional.

Example

```
*       Edit date as MM/DD/YYYY.
        MOVE input-date            TO WS-STRFLD-DATE.
        PERFORM 580-STRING-DATE-MMDDYYYY.
```

580-DATE-MMYYYY

The 580-DATE-MMYYYY routine strings the given date into the output area as MM/YYYY. No error indication is given for an invalid date.

Library STRRTNS

Required Input

Field	Type and length	Definition
WS-STRFMT-DATE	N 8	Given date.

Optional Output

Field	Type and length	Definition
WS-STRFLD-OUT	A 301	MM/YYYY format.
WS-STRFLD-PNTR	N 4	Optional.

Example

```
*       Edit date as MM/YYYY.
        MOVE input-date            TO WS-STRFLD-DATE.
        PERFORM 580-STRING-DATE-MMYYYY.
```

Deprecated Direct Inquiry Data Processing

IMPORTANT These routines are not recommended for user in application programs. Direct inquiry routines imply access without indexes, which can have a significant, negative affect on performance. Lawson recommends using the routines in "Database Input/Output Routines" on page 91.

800-OPEN and 860-READ

The following two routines differ from all the other find routines in that they do not use indexes or keys to inquire on records. Instead, they directly access a file to inquire on the records.

The 800-OPEN-<FileName>and the 860-READ-<FileName> routines combine to inquire on an entire file in the order in which records are stored. Use these routines only when reading an entire file and order is not significant. No key fields are used with either of these routines because records are retrieved directly from the file with no need to identify individual key field values. This has the advantage of skipping the extra call to the <Index> for each record inquiry.

800-OPEN<FileName> readies the database to start reading records at the physical beginning of the file. If a connection to the database has not been established, the routine establishes it here. This routine also populates memory to hold the file description and any other memory required for the table to communicate to the database.

Once the 800-OPEN-<FileName> routine prepares the file, use the 860-READ-<FileName> routine to retrieve the records from the file in the physical order in which they are stored. Each time 860-READ-<FileName> is called, it returns the next record in the file. When the routine reaches the end of the file, the next call to 860-READ-<FileName> returns <FileName>-NOTFOUND.

NOTE These routines are *not valid* while in transaction state.

Programming Example for Direct Inquiry Routines

The following example shows the direct inquiry routines inquiring on an entire file in its physical order.

```
 * Open GLMASTER, then read and process one record at a time.
 *
     PERFORM 800-OPEN-GLMASTER.
     PERFORM 860-READ-GLMASTER.
     PERFORM
         UNTIL (GLMASTER-NOTFOUND)
         PERFORM 999-PROCESS-GLM
         PERFORM 860-READ-GLMASTER
     END PERFORM.
```

In this example, the 800-OPEN-<FileName> call readies the file for reading. The PERFORM loop then uses the 860-READ-<FileName> statement to continue reading and processing more records until it reaches the end of the file.

Deprecated Database Aggregate Range Routines

880-INIT and 880-FIND Aggregate Range APIs

These two aggregate range routines are the older set of aggregate range routines that still exist for backwards compatibility. They work together to allow you to perform specific functions on one or more columns for a range of rows. The functions you can perform include average value for a column, maximum value for a column, minimum value for a column, and sum of a column.

You can use the aggregate range routines whenever you would otherwise use 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> to access a set of rows for the purpose of applying average, maximum, minimum, or sum functions against a column.

Use the aggregate range routines together.

Use the 880-INIT-DBAG-<Index> routine to initialize an aggregation request.

Use the 880-FIND-DBAG-<Index> routine to process an aggregation request against a data set.

Aggregate Request Format

In addition to using the DB-<KeyFieldName> and DBRNG-<KeyFieldName> fields as you do with other database inquiry routines, you use a set of Boolean fields to identify the columns in the table to be returned and the aggregate function to be applied. These fields are named using the following convention:

This field	Returns this value for the column designated by field name
DBAVG-<Index>-<FieldName>	Average
DBMAX-<Index>-<FieldName>	Maximum
DBMIN-<Index>-<FieldName>	Minimum
DBSUM-<Index>-<FieldName>	Sum

Where <Index> corresponds to and must match the Index that is specified in the 880-INIT-DBAG-<Index> and 880-FIND-DBAG-<Index> API calls, and <FieldName> identifies the column to be aggregated

You can request more than one column in a single API call by using the appropriate Boolean field several times with a different field name each time; but you may specify only one function for each call. Thus, in order to derive the average and the sum of any column, you must populate two separate requests and make two separate calls to the API.

Return Values

Unlike all other database APIs, the values returned by 880-FIND-DBAG-<Index> are not part of the regular file record area. In fact, nothing related to the aggregate range routines alters the contents of the file record area in any way. Because the SUM function can potentially return a value which overflows the boundaries of the field being derived, all return values are dimensioned to the maximum allowable field size of 18 digits. The number of decimal positions is inherited from the database field. The naming convention for these return value fields is:

DBAG-<Index>-<FieldName>

Using the Aggregate Range APIs

This section lists the steps and shows the source code you must use to use these aggregate range routines.

Programming Example

The following example uses the aggregate range routines to specify the SUM function.

```
***Using aggregate range routines to specify
***the SUM function

MOVE WS-COMPANY               TO DB-COMPANY.
MOVE WS-ACCOUNT               TO DB-ACCOUNT.
MOVE WS-FROM-SUB-ACCT         TO DB-SUB-ACCOUNT.
MOVE WS-THRU-SUB-ACCT         TO DBRNG-SUB-ACCOUNT.
MOVE GLMSET1-SUB-ACCOUNT      TO WS-DB-SUB-RNG.
PERFORM 880-INIT-DBAG-GLMSET1.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD)      TO TRUE.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD+1)    TO TRUE.
SET  DBSUM-GLMSET1-AMOUNT (WS-PERIOD-1)    TO TRUE.
PERFORM 880-FIND-DBAG-GLMSET1.
IF (GLMASTER-FOUND)
  MOVE DBAG-GLMSET1-AMOUNT (WS-PERIOD)
                           TO WS-CURR-PERIOD-TOTAL
  MOVE DBAG-GLMSET1-AMOUNT (WS-PERIOD + 1)
                           TO WS-NEXT-PERIOD-TOTAL
  MOVE DBAG-GLMSET1-AMOUNT (WS-PERIOD - 1)
                           TO WS-PREV-PERIOD-TOTAL.
```

Now, compare the code for the aggregate range routines to the code you would have to write to get the same functionality by using the 850-FIND-SUBRNG-<Index> and 860-FIND-NXTRNG-<Index> routines, shown in the following example.

```
*** Using SUBRNG routines to specify the SUM function

MOVE WS-COMPANY                    TO DB-COMPANY.
MOVE WS-ACCOUNT                    TO DB-ACCOUNT.
MOVE WS-FROM-SUB-ACCT              TO DB-SUB-ACCOUNT.
MOVE WS-THRU-SUB-ACCT              TO DBRNG-SUB-ACCOUNT.
MOVE GLMSET1-SUB-ACCOUNT           TO WS-DB-SUB-RNG.
PERFORM 850-FIND-SUBRNG-GLMSET1.
PERFORM UNTIL (GLMASTER-NOTFOUND)
  ADD GLM-AMOUNT (WS-PERIOD)       TO WS-CURR-PERIOD-TOTAL
  ADD GLM-AMOUNT (WS-PERIOD + 1)   TO WS-NEXT-PERIOD-TOTAL
  ADD GLM-AMOUNT (WS-PERIOD - 1)   TO WS-PREV-PERIOD-TOTAL
  PERFORM 860-FIND-NXTRNG-GLMSET1
END-PERFORM.
```

Accessing Database Records

STEPS **To access database records using the aggregate range routines**

1. Set the **DB-** key fields as you would for any database inquiry routine.
2. Express the ending value for the range key:

   `MOVE identifier TO DBRNG-fieldname.`
3. Identify the range key by moving the symbolic constant:

   `MOVE <IndexName>-<IndexFieldName> TO WS-DB-SUB-RNG.`

4. Initialize the aggregation request. This initializes the Boolean request fields and the return values:

   ```
   PERFORM 880-INIT-DBAG-<Index>.
   ```

5. Set Booleans describing which fields to aggregate and what function to apply. Remember that you may apply only one function for each call.

   ```
   SET DBAVG-<Index>-<FieldName> TO TRUE.

   SET DBMAX-<Index>-<FieldName> TO TRUE.

   SET DBMIN-<Index>-<FieldName> TO TRUE.

   SET DBSUM-<Index>-<FieldName> TO TRUE.
   ```

6. Submit the request:

   ```
   PERFORM 880-FIND-DBAG-<Index>.
   ```

7. Test for function validity by referencing the file FOUND switch:

   ```
   IF (filename-FOUND) ...
   ```

8. Un-reference the return values in the fields named **DBAG-<Index>-<FieldName>**:

   ```
   MOVE DBAG-<Index>-<FieldName> TO WS-RESULT.
   ```

880-FIND-DBAGOF

Name

880-FIND-DBAGOF-<Index>

Description

The 880-FIND-DBAGOF-<Index> API is similar to the 880-FIND-DBAG-<Index> API, with the difference that it provides protection against putting overflow values into the field storing the result returned by the aggregate API. If the aggregate result is too large for the field it would be stored in, a flag is set indicating an overflow occurred. You can then add code to control what the program does in the event of an overflow.

Using the API in a Program

You use the 880-FIND-DBAGOF-<Index> API similarly to the 880-FIND-DBAG-<Index> API. For information on the latter, see "880-INIT and 880-FIND Aggregate Range APIs" on page 225. The difference is that an overflow flag is set through the **WS-DB-AGG-OVERFLOW** field. You can check whether the value of this flag is "1" and, if it is, execute code that prevents updating the database with a value that is too large for the field it would be stored in.

Appendix B

Documentation Conventions and Support

Documentation Conventions

This document uses specific text conventions and visual elements.

Text Conventions

This	Represents
bold	A key name or function key name. For example, **Shift** is a key name and **Help (F1)** is a function key name. A value or command that must be typed exactly as it appears. A program or file name.
italics	A manual title or form name. An emphasized word or phrase. A placeholder for a user-defined value or variable.
(F1)–(F24)	A function key number. “Press **Help (F1)**” instructs you to press the key mapped for the **(F1)** function.
Key1+Key2	A key combination. “Press **Shift+FndNxt (F3)**” instructs you to press and hold down the **Shift** key and then press the **FndNxt (F3)** function key. Release both keys to complete the action.
[ ]	Optional parameters. Type none, one, or more of the parameters within the brackets. For example, the command: **qsubmit [-Un] [-j***JobQueue* **-d***Date* **-t***Time***]** *username jobname* means that you can type a specific job queue, date, or time, or you can omit these parameters.
[|]	Optional parameters. You can type only one of the parameters separated by a vertical line. For example, the command: **phraserpt[-n|t]** *BaseLanguage* **[***Translation***]** means that you can type either the **n** or **t** parameter; you cannot type both.
...	A parameter that can be repeated. For example, the command: **scrgen [-scxvV]** *productline* **[***systemcode* **[***programcode*...**]]** means that you can type any number of program codes.

Visual Elements

Before you start Information you need to know before you attempt the procedure or process.

IMPORTANT Important information to consider when you perform the procedure.

CAUTION Cautionary information about actions that involve a risk of possible damage to equipment, data, or software.

WARNING Warning information about actions that involve a risk of personal injury or irreversible destruction to the data or operating system.

Product Documentation

Lawson offers the following product documentation:

- Online help
- User guides and manuals
- Release notes and installation instructions

To find Lawson documentation, see the user interface or http://support.lawson.com. To obtain a login password and ID for the Support site, see your organization's Lawson contact or your Lawson client manager.

Support & Delivery

Lawson Support & Delivery (S&D) is available to all Lawson customers who are on maintenance support for Lawson products. See the *Global Support Manual* for the following information:

- What information to gather before you contact S&D
- How to contact S&D
- How S&D processes your request
- Which services are standard maintenance and which are billable

To find the *Global Support Manual*, see http://support.lawson.com. To obtain a login password and ID for the support web site, see your organization's Lawson contact or your Lawson client manager.

Documentation Contact

We welcome your questions or suggestions about Lawson documentation. Please send comments to documentation@lawson.com.

Appendix C

Related Lawson Documentation

Related Lawson Documentation

This guide is one in a series of Lawson administration guides. This series includes:

- *Lawson Administration: Server Setup and Maintenance*
- *Lawson Administration: Resources and Security*
- *Lawson Administration: LAUA Security*
- *Lawson Administration: Portal*
- *Lawson Administration: Jobs and Reports*
- *Lawson Administration: Translation*
- *System Utilities Reference Guide*
- *Lawson Administration: Data Access Using Oracle*
- For information on function keys and terminal configuration, formerly included in the *Lawson Administration Guide*, see the Lawson Knowledge Base.

In addition, Lawson offers the following product documentation:

- Online help
- Users guides and manuals
- Installation guides
- Release notes
- Integration guides for third-party products
- Enhancement and patch documentation

To find Lawson documentation, see the user interface or http://support.lawson.com. In addition to the Document Center at support.lawson.com, the Lawson support site includes the Lawson Knowledge Base. You can search the Knowledge Base for most product documentation as well as for special topics not covered in any manual. To obtain a login ID and password for the Support site, see your organization's Lawson contact or your Lawson client manager.

Index

5

7

8

9

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W