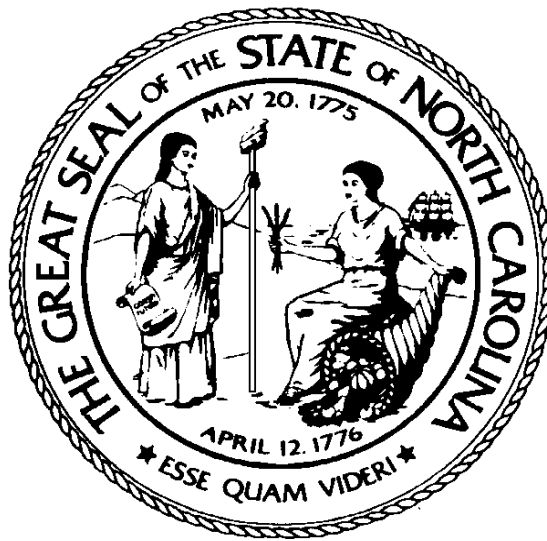


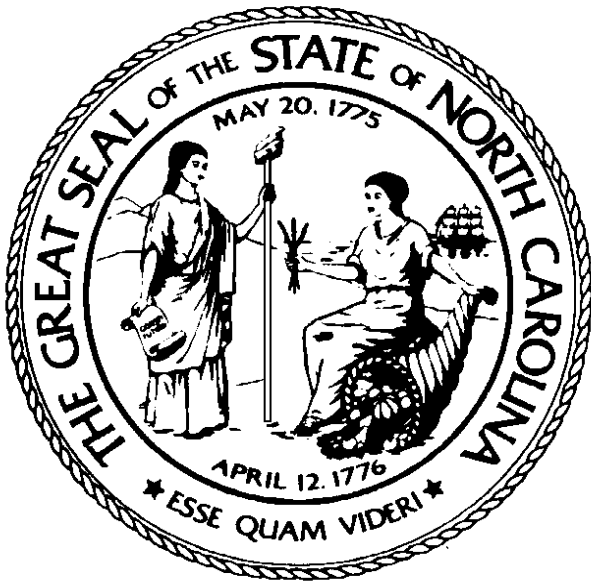
***Advanced Information
Expert Reporting
Training Course***



State of North Carolina

NC Accounting System

North Carolina Accounting System
Advanced Information Expert Reporting
Training Course



Robert L. Powell
State Controller
March 1, 2007

This training was prepared by
The Office of the State Controller

<http://www.ncosc.net>

Financial Systems Division
ASSISTANT STATE CONTROLLER
Julie Batchelor

NCAS TECHNICAL APPLICATIONS

Deborah Smith, Manager

NCAS INFORMATION ACCESS

Kelly Thomas, Manager

NCAS MODEL

Jim Macaulay, Manager

NCAS SUPPORT SERVICES

Terry Senter, Manager

TRAINING AND DOCUMENTATION

Owen Auman
Sue Crittenden
Sue Rader

Statewide Accounting Division

ASSISTANT STATE CONTROLLER

Wesley Ray

FINANCIAL RESEARCH & ANALYSIS

John Barfield, Manager

ACCOUNTING & FINANCIAL REPORTING

Anne Godwin, Manager

CENTRAL COMPLIANCE

Amber Young, Manager

TABLE OF CONTENTS

Course Overview	1
Overview	1
Audience	1
Length	1
Objectives.....	1
Course Map.....	2
Sections.....	2
Basic Testing Techniques	3
Overview	3
EXHIBIT Command.....	3
LIMIT Command.....	4
ABORT Command	4
EXIT Command.....	5
Recurring Fields and Looping Techniques.....	7
Overview	7
Recurring Fields	7
Subscripting Fields.....	8
FOR DO COMMAND	9
Creating a Recurring Field	10
WHILE DO Command.....	10
Advanced Data Access Techniques	13
Overview	13
Extraction Methods.....	13
Implicit Extraction	13
MATCH Command	15
Qualifying Items.....	17
NEXT RECORD Command.....	17
EXPLICIT Extraction	18
EXTRACT Command	18
ATTACH Command.....	19
READ NEXT Command	20
READ Command.....	21
POSITION Command.....	24
Creating an OUTPUT File.....	27
Overview	27
ATTACH Command.....	28
Moving Numeric Fields	30
INITIALIZE Command	32
Data Storage Techniques	35
Overview	35
DATASTORE Command.....	35
STORE Command.....	36
RECALL Command	37
EMPTY Command.....	38

Procedures and Subroutines	41
Overview	41
PROCEDURE Command.....	41
PERFORM Command.....	41
SUBROUTINE Command	42
PERFORM SUBROUTINE Command	44
Converting Alphanumeric to Numeric.....	47
QRG 1: OUTPUT Dataframes	51

Course Overview

Overview

The North Carolina Accounting System (NCAS) provides a complete information access environment. NCAS allows direct access using inquiry screens, a report management system (X/PTR) that distributes images of over 250 groups of NCAS reports, a Decision Support system (DSS) that is a client server based analytical tool for users, and Information Expert (IE) that is used as the primary reporting tool.

Information Expert is a software tool used for reporting. Most of the standard daily and monthly NCAS reports are produced using Information Expert. Information Expert is a flexible menu driven reporting tool used to create simple to complex reports. IE also has the capability to customize reports using a 4th generation language. Programmers or non-programmers can easily produce reports by using Information Expert.

Information Expert can also be used as a programming tool. Information Expert can create output datasets that can be downloaded to PC products such as Excel, Word or Access.

This course provides a practical, hands-on approach to using some of the advanced features of the Information Expert language.

Audience

Experienced NCAS IE users

Length

1 day

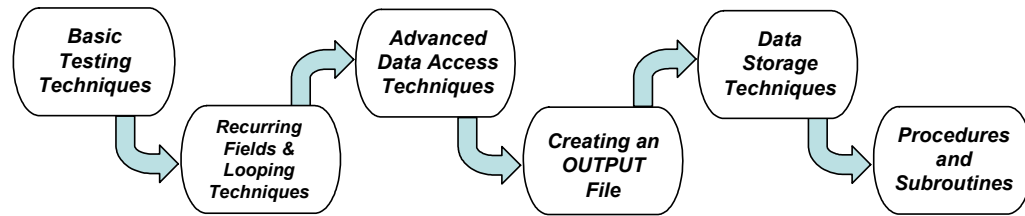
Objectives

Upon successful completion of this course, participants will be able to:

- Understand basic testing techniques
- Understand advance data access techniques
- Understand how to use data storage commands
- Understand how to use looping commands
- Understand how to use and create procedures and subroutines
- Understand how to create output datasets

NOTES

Course Map

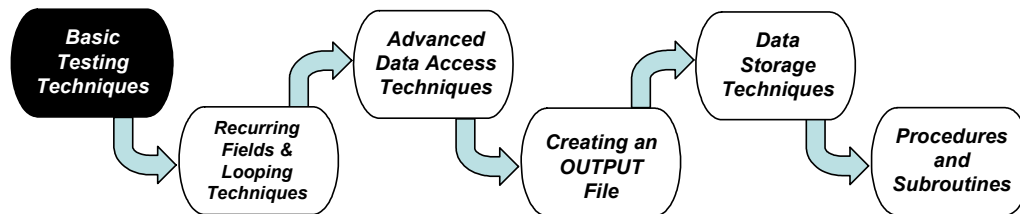


Sections

The *Advanced Information Expert Reporting* course is organized into the following sections:

- Basic Testing Techniques
- Recurring Fields and Looping Techniques
- Advanced Data Access Techniques
- Creating an OUTPUT File
- Data Storage Techniques
- Subroutines and Procedures

Basic Testing Techniques



Overview

This section explains some of the debugging commands available to assist Information Expert users when unidentified results or errors occur.

EXHIBIT Command

The EXHIBIT command is used to display values of fields from a dataframe or work items. The EXHIBIT command can be placed in the common section or the report request section. The EXHIBIT command displays the values of the items specified in the command for every record that executes the command. There is no limit on the number of EXHIBIT commands allowed in a report series. The displayed values are written to the JWSDBG dataset.

The format of the command is:

```
EXHIBIT {HEX/BOTH} 'text' item-name1 item-name2...
```

The HEX/BOTH option is used to specify how the information is to be printed. The default is display. To see unprintable characters, such as packed field, use the HEX or the BOTH option.

The 'text' option can be used to add a description to identify what item is being displayed and where in the program the command has been placed.

The item-name option is the item from the dataframe or a work item whose value is to be displayed. If the item is a work item, it must be defined before executing the EXHIBIT command that references it.

NOTES

Example:

```
EXHIBIT WK-AMOUNT  
EXHBIT HEX WK-AMOUNT  
EXHIBIT BOTH WK-AMOUNT  
EXHIBIT 'TRANS AMOUNT A' WK-AMOUNT
```

The results of these commands would be:

```
WK-AMOUNT: 000000007033190-  
WK-AMOUNT: 000000007033190D  
WK-AMOUNT:  
    00007310  
    0000039D  
TRANS AMOUNT A: 000000007033190-
```

LIMIT Command

The LIMIT command is used to control the number of records processed. The LIMIT command helps decrease costs and processing time while testing and debugging a report series. If the LIMIT command is placed in the common section, the command controls the number of input records processed in the common section. If the LIMIT command is used in a report request section, the command controls the number of extracted records processed for that report request.

The format of the command is:

```
LIMIT nnnn
```

nnnn = the number of records to process

Example:

```
INPUT C-DETAIL  
LIMIT 100
```

ABORT Command

The ABORT command is used to force a premature termination of the report series. A good practice is to check required variables to insure values are within prescribed ranges. The ABORT command can be used in the common section or the report request section. A unique abort code should be assigned to each ABORT command found in a report series. This allows easier identification as to why the program was terminated. If the ABORT command is used in the common section, it can be combined with the EXHIBIT command to display a message giving the error causing the abnormal termination.

The ABORT causes a system dump in the JWSDBG dataset. You are able to identify a user forced abort from an Information Expert system abort by abort code 4000. The user abort code then displays, allowing you to trace to the exact location in the program where the abort occurred.

The format of the command is:

NOTES

```
ABORT nnnn
```

nnnn = the abort code. This abort code can range from 1 to 1000.

Let's look at an example. In the example below, the variable PERIOD must be given a value in the run statement. If the variable is not set, a system abort occurs:

```
INPUT GLOPENYR
WORK PERIOD(20)
VARIABLE IS PERIOD
WK-AMOUNT = GL-PERIOD-ENDING-BALANCE/PERIOD
=====
*** ABEND HAS OCCURRED REPORT SERIES : C-TEST
-- ABEND "0000" OCCURRED DURING DATA EXTRACT PHASE
```

The abort information displays in dataset JWSDBG. The programmer needs to analyze the data from the system abort to determine the problem.

Code can be added to insure the variable PERIOD is set during run-time. Code can also be added to insure proper values are used. If the values are not properly set, a user abort can be forced.

```
INPUT GLOPENYR
WORK PERIOD(20)
VARIABLE IS PERIOD
IF PERIOD EQ 0
    EXHIBIT 'VARIABLE PERIOD HAS NOT BEEN SET ' PERIOD
    ABORT 100
END
IF PERIOD GT 13
    EXHIBIT 'VARIABLE PERIOD CONTAINS AN INVALID VALUE ' PERIOD
    ABORT 200
END

WK-AMOUNT = GL-PERIOD-ENDING-BALANCE/PERIOD
```

If the variable period was not set prior to execution, the results would be:

```
VARIABLE PERIOD HAS NOT BEEN SET: 00
=====
*** USER FORCED ABORT REPORT SERIES : C-IE04-TEST-5
-- ABEND "0100" OCCURRED DURING DATA EXTRACT PHASE
```

EXIT Command

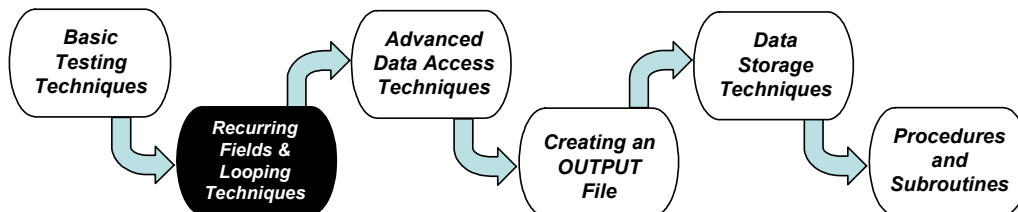
The EXIT command is used to bypass commands. The various formats of the EXIT command can be used to increase efficiency, eliminating execution of commands that are not necessary. The format of the command is:

```
EXIT
EXIT REPORT
EXIT COMMON
EXIT SUBROUTINE
EXIT PROC
```

NOTES

The EXIT command can be used to bypass remaining statements in a loop structure. Control is transferred to the END command, which then returns control to the beginning of the looping structure. The EXIT COMMON command can be used in the common section of the report series to bypass remaining common section commands. Control is passed to the end of the common section where automatic extraction occurs. EXIT REPORT command is used in the report request section and will transfer control to the end of the report request, bypassing remaining commands. The EXIT PROC and EXIT SUBROUTINE commands can be used in a procedure or subroutine to bypass remaining statements.

Recurring Fields and Looping Techniques



Overview

This section explains the Expert Language fields that are designated as recurring items. This section will review commands that are used to process recurring fields.

Recurring Fields

Recurring fields are items that have more than one occurrence. For example, in dataframe GLOPENYR, there is an item called GL-PERIOD-ENDING-BALANCE. If we look at the GLOPENYR dataframe in *System Administration List Dataframe* contents screen (SALF), we see the last column on the screen is OCCS. This stands for occurrences. The item GL-PERIOD-ENDING-BALANCE occurs 13 times, corresponding to the 13 period balance amounts. The system has established 13 separate occurrences of this item, all referenced by the same name.

```

D B S  INFORMATION EXPERT ----- LIST CONTENTS OF DATAFRAME GLOPENYR  JSALFD
FIND DATA NAME: gl-period-ending

```

ITEM / GROUP NAME	LVL	DISPL	TYPE	FLD SIZE	NBR DIG	DEC POS	DT CD	OCCS
FILLER	01	39	A	2				
GL-PERIOD-ENDING-BALANCE	01	41	P	8	15	2		13
GL-CURR-YEAR-LTD-BALANCE	01	145	P	8	15	2		
GL-CNTR-PROCESSING-CYCLE-NBR	01	153	A	3				
FILLER	01	156	A	4				
GL-CURR-YEAR-AGGREGATES-REC	00	0	A	160				
GL-COMPANY-ID	01	0	A	4				
GL-CHARGE-TYPE	01	4	A	1				
GL-ACCOUNT-ID	01	5	A	18				
GL-CENTER-ID	01	23	A	12				
FILLER	01	35	A	3				
GL-RECORD-ID	01	38	A	1				
FILLER	01	39	A	2				
GL-PERIOD-AGGREGATE-AMOUNT	01	41	P	8	15	2		14
FILLER	01	153	A	7				

```

ACTION: _____  1 Help  3 End  4 File Attr  6 Top  7 Pg Bwd  8 Pg Fwd

```

NOTES

When designing this dataframe, the programmer could have created 13 separate items, uniquely labeled with separate names, looking like this:

GL-PERIOD-ENDING-BALANCE-1	15P2
GL-PERIOD-ENDING-BALANCE-2	15P2
GL-PERIOD-ENDING-BALANCE-3	15P2
GL-PERIOD-ENDING-BALANCE-4	15P2
GL-PERIOD-ENDING-BALANCE-5	15P2
GL-PERIOD-ENDING-BALANCE-6	15P2 etc..

If you wanted to compute the six month average balance for the above 6 fields, your calculation would look like this:

$$\text{WK-AVG} = (\text{GL-PERIOD-ENDING-BALANCE-1} + ; \\ \text{GL-PERIOD-ENDING-BALANCE-2} + ; \\ \text{GL-PERIOD-ENDING-BALANCE-3} + ; \\ \text{GL-PERIOD-ENDING-BALANCE-4} + ; \\ \text{GL-PERIOD-ENDING-BALANCE-5} + ; \\ \text{GL-PERIOD-ENDING-BALANCE-6}) / 6$$

Subscripting Fields

To reference a particular occurrence of a recurring field, you use the occurrence as a subscript. To use the GL-PERIOD-ENDING-BALANCE for period 1, the reference to the item would be as follows:

	GL-PERIOD-ENDING-BALANCE/1
For period 2:	GL-PERIOD-ENDING-BALANCE/2
For period 3:	GL-PERIOD-ENDING-BALANCE/3

To use a subscript, place a slash (/) after the item name, followed by the subscript. There are no spaces between the item name, slash and subscript. The subscript will always be a number or a numeric item. The subscript can never take on a value greater than the number of occurrences defined or less than 1. The results may be unpredictable or an abnormal termination may occur if the subscript goes outside these bounds.

To perform the six-month average calculation above on this recurring item, the calculation would look like this:

$$\text{WK-AVG} = (\text{GL-PERIOD-ENDING-BALANCE/1} + ; \\ \text{GL-PERIOD-ENDING-BALANCE/2} + ; \\ \text{GL-PERIOD-ENDING-BALANCE/3} + ; \\ \text{GL-PERIOD-ENDING-BALANCE/4} + ; \\ \text{GL-PERIOD-ENDING-BALANCE/5} + ; \\ \text{GL-PERIOD-ENDING-BALANCE/6}) / 6$$

If a recurring item is used in a calculation, you must include the subscript. If a recurring item is used in a PRINT or LIST command, a subscript is optional. If the subscript is not used, all occurrences are displayed.

Recurring items are NOT displayed in Expert Reporting (10-step).

FOR DO COMMAND

NOTES

The FOR DO command is a command that creates a looping structure. A looping structure is a series of commands that is executed until a condition is met. The FOR DO command must be terminated by a stand-alone END command. The FOR DO command can occur in the common section or report request. There are no limits on the number of FOR DO commands that may be used.

All commands within the FOR DO and END command lines are executed each time the loop is executed. The EXIT command can be used to skip to the end of the loop without executing the remaining commands in the looping structure. The FOR DO command is a command that can also be used to automate subscripting. The format of the FOR command is:

```
FOR item FROM begin TO end BY increase DO
    ...IE commands
END
```

item is a work field that will be used as the subscript.
begin is the initial value to begin the subscript
end is the largest value of the subscript
increase is the amount to increment the subscript

Let's look at the above example. The calculation uses the first 6 occurrences to perform the average calculation. We can use a FOR DO loop to complete this calculation:

```
WORK WK-SUB(S)
FOR WK-SUB FROM 1 TO 6 BY 1 DO
    WK-AVG = WK-AVG + GL-PERIOD-ENDING-BALANCE/WK-SUB
END
WK-AVG = WK-AVG / 6
```

The work field WK-SUB has been defined with an LTD of S (subscript). This is a special numeric type used for subscripts. A subscript can also be defined as any of the numeric item types: N,O,P or Q. The above FOR DO command would initialize WK-SUB at 1, increase WK-SUB by 1 until WK-SUB is greater than 6. All commands within the looping structure would be executed 6 times.

The BY option is optional, the default increases the subscript by 1.

The FOR DO loop can also be decremented. We could also code the same example this way:

```
WORK WK-SUB(S)
FOR WK-SUB FROM 6 TO 1 BY -1 DO
    WK-AVG = WK-AVG + GL-PERIOD-ENDING-BALANCE/WK-SUB
END
WK-AVG = WK-AVG / 6
```

NOTES

Creating a Recurring Field

A recurring field can be created in a report series using a WORK command. After the length, type and decimal places or date format (LTD), the maximum number of occurrences can be specified. For example:

```
WORK WK-PER-DESC(3A)/12
```

This work field now has 12 occurrences. To load this field we could include this code in a report series (only the first 6 are initialized in the code below) :

```
FIRST TIME DO
  WK-PER-DESC/1 = 'JUL'
  WK-PER-DESC/2 = 'AUG'
  WK-PER-DESC/3 = 'SEP'
  WK-PER-DESC/4 = 'OCT'
  WK-PER-DESC/5 = 'NOV'
  WK-PER-DESC/6 = 'DEC'
END
```

Then to use these fields, we could use the FOR DO loop in the report request:

```
WORK WK-SUB(S)
FOR WK-SUB FROM 1 TO 6 DO
  PRINT AT 1 'PERIOD: ' WK-PER-DESC/WK-SUB;
  AT 15 GL-PERIOD-ENDING-BALANCE/WK-SUB
END
```

WHILE DO Command

Another command that builds a looping structure is the WHILE DO command. This command executes the commands within its looping structure until a condition is met. The WHILE DO command must be terminated by a stand-alone END command. The format of the WHILE DO command is:

```
WHILE condition DO
  ...IE COMMANDS
END
```

Before the WHILE looping structure is executed, the condition is tested. If the condition is true, the IE commands within the looping structure are executed. If the condition is false, control is passed to the next command after the END statement.

The conditions are built similar to a condition on an IF statement. Conditional operators of EQ, NE, LT, LE, GT, and GE can be used. Simple or compound conditions are available on the WHILE DO command.

Let's look at an example:

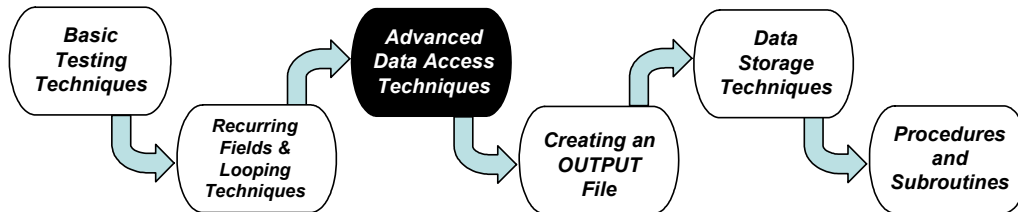
NOTES

```
WORK WK-SUB(S)
WK-SUB = 1
WHILE WK-SUB LE 6 DO
    WK-AVG = WK-AVG + GL-PERIOD-ENDING-BALANCE/WK-SUB
    WK-SUB = WK-SUB + 1
END
```

In the above example, we completed the same process the FOR DO command did in the previous example. We initialized a work field WK-SUB. The WHILE DO looping structure will execute 6 times, until WK-SUB is greater than 6.

When using the WHILE DO looping structure, insure your condition will be met. If the condition is not met, your report series could perform an infinite loop, terminating when time has run out. This can be VERY costly. A good practice when first learning to use the WHILE DO looping command is to use the EXHIBIT command to display the items in the condition, to insure they are changing according to the program.

Advanced Data Access Techniques



Overview

This section explains the Expert Language commands used for data access. The difference between implicit (automatic) extraction and explicit extraction will be discussed. The user will create a report series using these advanced techniques, submit the report to prepare, and run the report.

Extraction Methods

To create a report using Information Expert, data is extracted from the primary dataframe, any secondary dataframes accessed in the common section or work fields created in the common section. Extract records are created for each report request based on the items used in the individual report requests. There are two methods to extract data, implicitly (or automatically) or explicitly. Implicit extraction requires no command telling Information Expert to create an extract record – it's automatic. Explicit extraction requires a command to tell Information Expert to create an extract record - it is NOT automatic.

Implicit Extraction

There are two commands that can be used that perform implicit extraction - the INPUT and MATCH commands.

INPUT COMMAND

The INPUT command specifies the primary dataframe to be processed by the report series. If used, it *MUST* be the first command in the report series. There can only be one INPUT command in a report series.

The format of the command is:

INPUT dataframe

The dataframe name is from one to eight characters. The dataframe must already exist on the data dictionary.

Here is an example of the INPUT command:

INPUT GLOPENYR

NOTES

The INPUT command reads the first record of the dataframe and presents the data to the common section. If the dataframe is structured, the dataframe reads all the records necessary to build the structure and presents them to the common section. The common section is then processed with the presented data. At the end of the common section, IE evaluates the first report request to determine if the record is needed for reporting. If the record is needed, any fields used in the report request are automatically written or "extracted" to a report extract file. This evaluation process is executed for each report request. Fields written to the extract file can be different for each report request. Each record in the extract file for a particular report request will be the same, but the extracted records can be completely different for other report requests within the same report series. Reports are generated using the extracted records for each report request.

Let's look at an example:

```
INPUT GLOPENYR
RUN-TIME SELECT GL-COMPANY-ID
WORK PERIOD (20)
VARIABLE IS PERIOD
REPORT R1
WHEN CHANGE OCCURS IN GL-ACCOUNT-ID
      LIST  GL-ACCOUNT-ID;
           GL-ACCT-DESCRIPTION
END
REPORT R2
EXCLUDE GL-CENTER-ID '999999999999'
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      GL-ACCT-DESCRIPTION;
      GL-CENTER-ID;
      GL-PERIOD-ENDING-BALANCE/PERIOD
```

The extract record for report R1 would include 2 items:

```
GL-ACCOUNT-ID
GL-ACCT-DESCRIPTION
```

The extract record for report request R2 would include 6 items:

```
GL-COMPANY-ID
GL-ACCOUNT-ID
GL-ACCT-DESCRIPTION
GL-CENTER-ID
GL-PERIOD-ENDING-BALANCE
PERIOD
```

The INPUT command continues to read all records from the input dataframe, using the same process to evaluate and build the extract records. The extract file is then read, using the extracted records for each report request, performing the calculations and logic in each report request to generate the report or the required process.

Remember, the extraction process is automatic when the INPUT command is used.

MATCH Command

NOTES

The MATCH command is used to compare two or more dataframes. The MATCH and INPUT command cannot be used in the same report series. If the MATCH command is used, it *MUST* be the first command in the report series. The dataframes specified on the MATCH command *MUST* have the exact same sequence defined in the ORGANIZED BY. The exact same sequence means the exact same item names in the exact same order.

The format of the command is:

```
MATCH dataframe1 dataframe2...
```

When the MATCH command is issued, IE reads one record from each dataframe. If the dataframe is structured, IE reads all the records belonging to the structure. After the record information is retrieved, IE compares the sequences of the two sets of records. If all the values in the items on the ORGANIZED BY match, the system presents each of the records from the separate dataframes to the common section for processing. If the values in the items on the ORGANIZED BY do not match, the system presents the record from the dataframe with the lowest sequence. Items in the other dataframes that did not match to the lowest sequence are initialized.

Let's look at an example:

```
GLOPENYR  ORGANIZED BY  GL-COMPANY-ID;  
           GL-ACCOUNT-ID;  
           GL-CENTER-ID  
  
GLPTDJE   ORGANIZED BY  GL-COMPANY-ID;  
           GL-ACCOUNT-ID;  
           GL-CENTER-ID
```

These two dataframes are valid in a MATCH command since the two dataframes were defined with the exact same ORGANIZED BY. To view the organized by for a dataframe, you can use the *SALF command (System Administration, List Dataframes)*. Refer to your IE03 manual (*Basic Information Expert Reporting*) for more details.

Logic can be added to test if the match to the dataframes is successful. One test could check if both dataframes are present for the same values in the ORGANIZED BY. To perform this check, add the following commands:

```
IF ALL.PRESENT  
   ...add IE commands here  
END
```

Another test could check to see which dataframe is missing. That command would be as follows:

```
IF GLOPENYR.PRESENT AND GLPTDJE.NOT-PRESENT  
   .... add IE commands here  
END
```

NOTES

In our example, the GLOPENYR dataframe contains the period ending balances and the account description. The dataframe GLPTDJE contains the period to date detail posting transactions. Assume there is a request for a report listing the detail transactions for an accounting distribution along with the beginning and ending balances for August 2005 (period 2, Fiscal Year 2006). We would want to match these two dataframes retrieving the beginning balance and ending balances from the GLOPENYR dataframe, and list all the detail transactions from the GLPTDJE dataframe. We might also want to calculate an ending balance from summarizing the detail posting transactions to insure the ending balance (and our report) is correct. The following would be how the report series would be coded:

```
MATCH GLOPENYR GLPTDJE
REPORT R1
SELECT GL-ENTRY-RELATIVE-PERIOD-IND 02
SELECT GL-FISCAL-YEAR-OF-ENTRY '2006'
WHEN CHANGE OCCURS IN GL-CENTER-ID
  PRINT AT 1 'BEGINNING BALANCE';
    AT 50 GL-PERIOD-ENDING-BALANCE/01
END
IF GL-DEBIT-CREDIT-CODE LT '40'
  WK-AMOUNT(15p2) = GL-POSTING-ENTRY-AMOUNT
ELSE
  WK-AMOUNT = GL-POSTING-ENTRY-AMOUNT * -1
END
TOTAL WK-AMOUNT BY GL-CENTER-ID
LIST AT 5  ALL      GL-COMPANY-ID;
  AT 10  ALL      GL-ACCOUNT-ID;
  AT 30  ALL      GL-CENTER-ID;
  AT 50          WK-AMOUNT HEADING IS 'POSTING,AMOUNT'
WHEN CHANGE SENSED IN GL-CENTER-ID
  PRINT AT 1 'ENDING BALANCE';
  AT 50 GL-PERIOD-ENDING-BALANCE/02
  WK-ENDING-BALANCE(15p2) = GL-PERIOD-ENDING-BALANCE/01 + ;
    GL-CENTER-ID.WK-AMOUNT
  PRINT AT 1 'CALCULATED BALANCE';
  AT 50 WK-ENDING-BALANCE
  ADVANCE 3
END
```

When the MATCH command is used in a report series, the system automatically extracts data from the input dataframes. The extract process is the same as that used in with the INPUT command. As with the INPUT command, the automatic extraction is implicit, there does not have to be a separate command issued to create the extracted record. Automatic extraction occurs after the last command in the common section is executed.

Qualifying Items

NOTES

When two or more dataframes are referenced in the same report series, there is a chance that there are identical items. In the above report series, GLOPENYR and GLPTDJE are the two dataframes that were used. Some of the fields that are identical in both dataframes are:

```
GL-COMPANY-ID  
GL-ACCOUNT-ID  
GL-CENTER-ID  
GL-CLASS-CODE
```

In order to indicate you want the values for the items from a specific dataframe, you can qualify the items. To qualify an item, you attach the dataframe name to the item name, separated by a colon. To use items from the GLOPENYR dataframe, you could qualify the items as follows:

```
GLOPENYR:GL-COMPANY-ID  
GLOPENYR:GL-ACCOUNT-ID
```

Or, to use the items from the GLPTDJE dataframe, qualify the items as follows:

```
GLPTDJE:GL-CENTER-ID  
GLPTDJE:GL-CLASS-CODE
```

If the item is located in multiple dataframes and the records are present, the values are usually the same, therefore there is no reason to qualify the item names. If records are not present in one dataframe, you may want to specify which dataframe to display the values from by qualifying the item names.

NEXT RECORD Command

To prevent a record from automatically extracting, the NEXT RECORD command can be used. When the NEXT RECORD command is encountered in the common section, IE bypasses further commands for the current input record. Since the current input record did not pass to the end of the common section where the automatic extraction occurred, that record is not extracted. When the NEXT RECORD command is encountered in a report request, IE bypasses further processing of the current extract record and retrieves the next extract record.

Caution should be used when using the NEXT RECORD command and control break processing (WHEN CHANGE SENSED/OCCURS). If the record that is being eliminated is also the record that senses the change, the control break may not be recognized. Results could be unpredictable.

The following is an example of the NEXT RECORD command:

```
IF WK-AMOUNT EQ 0  
  NEXT RECORD  
END
```

NOTES

EXPLICIT Extraction

EXTRACT Command

There are other commands that are available in Information Expert that can read records from a dataframe. These commands do not automatically build and write the extract records to process the report requests. To extract records when using these other commands, the EXTRACT command can be used to force an extract record to be built. The format of the EXTRACT command is:

```
EXTRACT DATA FOR report1 report2.....
```

When the EXTRACT command is encountered, values for fields set at that time are used to build the extract record for the report request specified on the command. The EXTRACT command can be used with the INPUT and MATCH commands that perform automatic extraction. If the EXTRACT command is used in addition to the commands that perform automatic extraction, multiple records will be extracted.

Let's look at an example:

```
INPUT GLOPENYR
SELECT GL-COMPANY-ID '1401'
WK-RCC = GL-CENTER-ID(5 4)
IF WK-RCC = '2000' AND GL-PERIOD-ENDING-BALANCE/1 NE 0
  EXTRACT DATA FOR R1
END
REPORT R1
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      GL-CENTER-ID
```

The intent of this report series is to list accounting distributions for responsibility center 2000 that have a non zero balance. There is an explicit EXTRACT command. Since the INPUT command was used, IE also performs the implicit automatic extraction. Two extract records are created for each input record. To prevent duplicate records from being extracted, use the NEXT RECORD command as the LAST command in the common section.

```
INPUT GLOPENYR
SELECT GL-COMPANY-ID '1401'
WK-RCC = GL-CENTER-ID(5 4)
IF WK-RCC = '2000' AND GL-PERIOD-ENDING-BALANCE/1 NE 0
  EXTRACT DATA FOR R1
END
NEXT RECORD
REPORT R1
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      GL-CENTER-ID
```

Generally, when using the INPUT or MATCH commands with the EXTRACT command, you have a NEXT RECORD as the last command in the common section to prevent duplicate records from extracting.

ACTIVITY

NOTES

SCENARIO

You have received a report request to review posting relationships. The accounting staff wants to know which accounting distributions have no posting transactions. You will create a report series using the MATCH command to complete this assignment.

1. In Source Management, go to *SMCR* and create a report series C-GL-MATCH-EXERCISE.
2. Use GLOPENYR and GLPTDJE dataframes. GLOPENYR contains the General Ledger Balances. GLPTDJE contains the actual posting transactions.
3. Add logic to extract only records that have no posting transactions. Exclude GL-CENTER-ID of '999999999999'.
4. Use the LIST command to print the following fields:

GL-COMPANY-ID
GL-ACCOUNT-ID
GL-CENTER-ID
5. CHECK the report series, fixing any errors.
6. SAVE and SUBMIT the report series to prepare.
7. Go to *Report Viewing (RVLS)* to insure the preparation was successful.
8. Go to *Job Submission (JSRU)* to run the report series.
9. View the report results using *Report Viewing (RVLS)*.

ATTACH Command

The ATTACH command tells Information Expert that you want to perform user controlled I/O on the dataframe specified in the command. The format of the command is:

ATTACH dataframe mode

The different modes of the ATTACH command are RETRIEVE, CREATE and EXTEND. RETRIEVE is the default mode. The RETRIEVE mode is used with one of the user controlled commands to read a dataframe. There are two user controlled commands that can be used to read a dataframe: READ NEXT and READ.

NOTES

READ NEXT Command

The READ NEXT command can be used to control the input and extraction process. The READ NEXT command sequentially reads a dataframe. The READ NEXT command can be used in the common section or in a report request. The dataframe must be ATTACHED using RETRIEVAL before being used on a READ NEXT command. The ATTACH command must be in the same section where the READ NEXT occurs. There are two forms of the READ NEXT command:

```
READ NEXT dataframe  
READ NEXT dataframe UNTIL END
```

The first form of the READ NEXT command sequentially reads one record of the dataframe. If all the records of the dataframe are to be read, looping logic has to be included.

When you use the READ NEXT command, you are in complete control of the dataframe processing. This means that all dataframes are processed as though they are non-structured dataframes. If there are multiple records in a dataframe, it is your responsibility to determine which record you have before you can use items in that record. Sequence break processing is not performed in the common section when using the READ NEXT command. If you required sequence break processing in the common section, you have to add your own IE code to handle break processing.

FIRST TIME DO and LAST TIME DO commands are also not available when using the READ NEXT commands. It is very important that you know when the last record has been read. If you read past the end of the file, the program will abnormally terminate.

There is no automatic extraction when using the READ NEXT commands. If a record is required to be extracted for a report request, the EXTRACT command is required.

There are variables that can be checked to determine if the last record has been read to handle end of file processing. These variables are LAST-RECORD and NOT-LAST-RECORD. These variables are concatenated to the dataframe name. These variables can be used in a logic statement to stop processing after the last record has been read.

Let's put all this together in a report series. To perform user controlled I/O on the GLPTDJE dataframe using the READ NEXT command, the following commands could be used, incorporating the WHILE command from the previous section:

NOTES

```
ATTACH GLPTDJE RETRIEVE
WORK READ-IND (1A)
READ-IND = 'Y'
WHILE READ-IND = 'Y' DO
    READ NEXT GLPTDJE
    IF GLPTDJE.NOT-LAST-RECORD
        EXTRACT DATA FOR R1
    ELSE
        READ-IND = 'N'
    END
END
REPORT R1
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      GL-CENTER-ID
```

The second form of the READ NEXT command handles the end of file processing for you. The format READ NEXT UNTIL END sequentially reads the dataframe until end of file. If the READ NEXT UNTIL END is replaced in the above code, the report series would look like this:

```
ATTACH GLPTDJE RETRIEVE
READ NEXT GLPTDJE UNTIL END
    EXTRACT DATA FOR R1
END
REPORT R1
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      GL-CENTER-ID
```

Notice that the EXTRACT command was used. When this command is executed, the extract record for the desired report is created. If the EXTRACT command was removed, no records would have been extracted for the report request.

READ Command

The READ command is similar to the RELATE command. It is a user controlled I/O command so it must be specified on an ATTACH command with RETRIEVAL mode. The READ command randomly accessed a datafile. The READ command can be used in the common section or in a report request. The ATTACH and READ commands must be placed in the same section.

With the READ command, you must provide the exact key for the dataframe record that you wish to read. To view the key to a record, use the SALF command. (Refer to your IE03 handout discussion on the *System Administration List Dataframes (SALF)*). When using the RELATE command, values for the items in the key fields were specified. For the READ command, the whole key must be created. The format of the READ command is:

```
READ dataframe KEY IS key
```

NOTES

Let's look at an example. A report has been requested to list posting transactions and include the account description. Let's complete this assignment using user controlled I/O commands. Dataframe GLPTDJE can be used for the posting transactions. Dataframe GLACCTPL is a dataframe that contains the account record where the account description is located. If we look at this dataframe on *SALF*, we see there is only one record. If we select the only record in the dataframe, we see the "KEY IS" statement.

```

D B S  INFORMATION EXPERT  -----  DISPLAY RECORD ATTRIBUTES  JSALFA

DATAFRAME NAME: GLACCTPL

RECORD NAME: GL-ACCOUNT-POLICY-RECORD

LEVEL =  GL-ACCOUNT-ID

IDENTIFIED BY:  GL-RECORD-ID  EQ  'B'  AND  GL-CHARGE-TYPE  EQ  'B'

KEY IS:  GL-COMPANY-ID  'B'  GL-ACCOUNT-ID  '          '  'B'  '  '

ACTION: _____  PF1 Help  PF3 End
    
```

We see the key length is 41 characters and is made up of the following:

GL-COMPANY-ID	04 characters	company
B	01 character	charge type
GL-ACCOUNT-ID	18 characters	account
15 blanks	15 characters	center / year sequence
B	01 character	record type
2 blanks	02 characters	Filler
TOTAL	41 chacters	

We need to build a work field that contains the exact 41-character key to match this record's key. The LTD for the key would be 41A. Using concatenation, build the key with items and fixed fields. This would be an example of the key:

```

ATTACH GLACCTPL RETRIEVE
WK-ACCOUNT-KEY(41A) = GL-COMPANY-ID . 'B' . GL-ACCOUNT-ID . 15' ' . 'B' . 2' '
READ GLACCTPL KEY IS WK-ACCOUNT-KEY
    
```

To insure the key is built successfully and a description record is located, a check should be made to insure the READ worked properly. To check to see if a record is located, .PRESENT and .NOT-PRESENT is used concatenated with the DATAFRAME name. Let's add this to our code:

NOTES

```
ATTACH GLACCTPL RETRIEVE
WK-ACCOUNT-KEY(41A) = GL-COMPANY-ID . 'B' . GL-ACCOUNT-ID . 15' . 'B' . 2'
READ GLACCTPL KEY IS WK-ACCOUNT-KEY
IF GLACCTPL.PRESENT
    WK-ACCOUNT-DESCRIPTION = GL-ACCT-DESCRIPTION
ELSE
    WK-ACCOUNT-DESCRIPTION = 'NOT FOUND'
END
```

So, if we combine the READ logic with the READ NEXT logic, this is what the report series would look like:

```
ATTACH GLPTDJE RETRIEVE
READ NEXT GLPTDJE UNTIL END
    ATTACH GLACCTPL RETRIEVE
    WK-ACCOUNT-KEY(41A) = GL-COMPANY-ID . 'B' . GL-ACCOUNT-ID . 15' . 'B' . 2'
    READ GLACCTPL KEY IS WK-ACCOUNT-KEY
    IF GLACCTPL.PRESENT
        WK-ACCOUNT-DESCRIPTION = GL-ACCT-DESCRIPTION
    ELSE
        WK-ACCOUNT-DESCRIPTION = 'NOT FOUND'
    END
    EXTRACT DATA FOR R1
END

REPORT R1
LIST  GL-COMPANY-ID;
      GL-ACCOUNT-ID;
      WK-ACCOUNT-DESCRIPTION HEADING IS 'ACCOUNT DESCRIPTION';
      GL-CENTER-ID
```

EXERCISE

1. Look at the above code. Is the READ command efficient? Explain why or why not:

2. What changes can be made to make the code more efficient?

NOTES

SCENARIO

In this scenario, you will use the READ NEXT and READ commands. You have been given a task to improve the efficiency of the preceding code.

1. In Source Management, go to *SMCR* to create a new report series C-IE04-READ-EXERCISE.
2. Add the above code, changing it to improve the efficiency.
3. CHECK your report series to insure you have no errors.
4. SAVE and SUBMIT the report series to prepare.
5. Go to the *Report Viewing List screen (RVLS)* to view your preparation results.
6. Use *Job Submission (JSRU)* to run the report series.
7. Use *Report Viewing (RVLS)* to view your results.

POSITION Command

The POSITION command is used to point to a location on a dataset. If you do not have a way to create the complete key, you could use a partial key and use the POSITION command. The format of the POSITION command is:

POSITION dataframe KEY IS key

The dataframe specified on the POSITION command must be used in an ATTACH command with RETRIEVAL mode. The ATTACH and POSITION commands must be located in the same section of code. The POSITION command does not read a record. The POSITION command points to a record closest to the specified key. If a record is not found that is an exact match for the key, the POSITION command points to the next higher record. To read the record, issue a READ NEXT command.

The key should be the same length as the key in the dataframe. If a partial key is used, supply as much of the key as possible when building the key.

Let's look at an example:

During nightly production there is a dataset that is built called the daily detail file. This daily detail file contains the detail posting transactions for each night. This daily detail file is combined with the previous detail posting transactions to create a fiscal year file. This annual detail file is used to generate the DAPG67 report. The detail file is created from the summarized transactions posted on the General Ledger GWP12 file. The detail from the summarized transactions reside on the Financial Controller Audit file (FCFMAUD).

NOTES

```
D B S INFORMATION EXPERT ----- DISPLAY RECORD ATTRIBUTES JSALFA

DATAFRAME NAME: C-FCAUD
RECORD NAME: C-FC-DETAIL-RECORD
LEVEL = C-FC-TRANS-TYPE
IDENTIFIED BY: C-FC-RECORD-TYPE EQ '02'
KEY IS: '02' C-FC-AUDIT-ID '000000000'

ACTION: _____ PF1 Help PF3 End
```

The key length is 21 characters. We have to construct a key using Financial Controller audit identifier. When a transaction is summarized, the audit identifier is placed in the General Ledger source code field.

If we look at the actual audit file, we can review how the key looks:

```
02*OGH013707000000000 AF3000000000342176P02
02*OGH013707000000001 AF3000000000342176P03 1200006309
02*OGH013707000000002 AF3000000000342176P03 1200006309
```

The first 2 characters are always '02'. The next 10 characters is the audit identifier. Notice there are 3 records with the same first 12 characters. The next 10 positions are the sequence number. If we look at the Financial Controller documentation, we see the 02 record type (in bold) is the summarized entry. Since that is what we already have in the GWP12, we do not want that record. We want the 03 record types that contain the detail records.

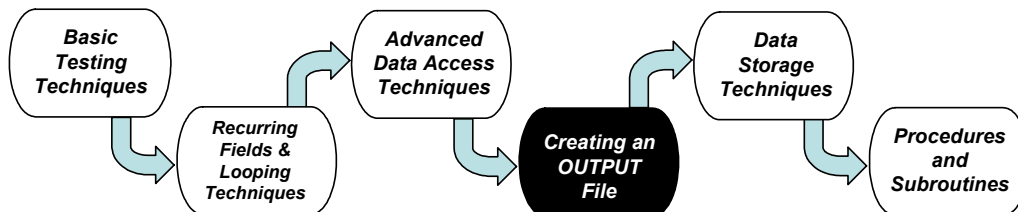
Let's look at the code to build the detail transactions:

NOTES

```
INPUT C-GM331
ATTACH C-FCAUD
ATTACH C-DETAIL CREATE
WORK WK-GL-KEY(21A)
WORK WK-FC-KEY(21A)
WK-GL-KEY = '02' . C-GL-SOURCE-CODE . 9'0'
* Position at the start of the key, read the first record
POSITION C-FCAUD KEY IS WK-GL-KEY
READ NEXT C-FCAUD
* Build key from FC file
WK-FC-KEY = '02' . C-FC-AUDIT-ID . 9'0'
* Compare keys
WHILE WK-FC-KEY = WK-GL-KEY DO
    IF C-FC-TRANS-TYPE EQ '01' OR '03'
        PERFORM WRITE-OUTPUT-RECORD
    END
    READ NEXT C-FCAUD
    WK-FC-KEY = '02' . C-FC-AUDIT-ID . 9'0'
END
```

After the POSITION command is issued, a READ NEXT command is executed to actually read the record that the POSITION command targeted. We issue a WHILE loop to continue to read records until the General Ledger key and Financial Controller key are different. Notice the sequence number in the key is always set to 9'0' (nine zeros). We actually have a partial key since the sequence number is not present on the General Ledger file. For comparison purposes, we create a key that is common between the two files.

Creating an OUTPUT File



Overview

Output files can be created using Information Expert. To create an output file, a dataframe must be selected. The dataframe selected must have the capability to write records. If the dataframe is a type LOGICAL, you must verify the logical interface module (LIM) has the capability to execute the write function. Dataframe types of VSAM and SEQUENTIAL use a generalized output module that has the capability to execute the write function. Choose a dataframe that has a record size that meets your needs. A list of dataframes that can be used to create a dataset can be found in the back of this manual (QRG 1).

When you select a dataframe, the DDNAME option specified on the dataframe definition directs IE to the physical dataset to which the records will be written. This name corresponds to the JCL dataset definition that is required to point to the physical dataset to be written. The DDNAME can also be viewed by displaying the file attributes for a dataframe using *System Administrator Dataframe Directory List (SALF)*.

```
D B S  INFORMATION EXPERT ----- SHOW FILE ATTRIBUTES  JSALFF

DATAFRAME NAME: C-80BYTE   TYPE: SEQUENTIAL   DDNAME: C80BYTE
RECORD FORMAT: FB        RECORD LENGTH: 80   BLOCKSIZE: 0
KEY LENGTH: 0           KEY LOCATION: 0
DSNAME: N/A

ACTION: _____   PF1 Help   PF3 End   PF8 List Records
```

NOTES

In the example of the preceding dataframe C-80BYTE, the DDNAME specified is C80BYTE. The JCL statement that must be added to our RUN-JCL would be:

```
//C80BYTE DD DSN=datasetname,DISP=SHR
```

Besides the DDNAME, other attributes of the dataset can be found. Each dataframe has the type, record length and block size that corresponds to the type of dataset it can produce. If the type is SEQUENTIAL or SEQ, a sequential file is created. If the type is VSAM, a VSAM file, usually a KSDS or ESDS file is created. If the type is LOGICAL or LOG, check the logical interface module (LIM) to determine the file type.

The physical dataset must have the corresponding record lengths and block size with the dataframe. If there are variable record sizes, the dataset must be defined to accept the largest record size. You should create your physical dataset and change the RUN-JCL member before trying to execute a report series that will create the output file.

ATTACH Command

Before we can write to a dataset, the dataframe must be ATTACHed. There are two ATTACH options that can be used when writing records to an output file – CREATE and EXTEND. Here is an example of the ATTACH command with the two options:

```
ATTACH C-80BYTE CREATE  
ATTACH C-80BYTE EXTEND
```

The CREATE option of the ATTACH command initializes the dataset and erases any records currently in the physical output file. The EXTEND option appends the records to the physical output file.

Whenever the EXTEND option is used, the disposition of the dataset in the JCL must be MOD. If the disposition of the dataset is OLD or SHR, the records written in the last report request processed with a WRITE command are the only records that remain in the output file.

The WRITE command is used to specify which records of a selected dataframe to write to an output file. To write to a dataframe specified on the ATTACH command, move the data to fields in a record on the dataframe. When the WRITE command is issued, it will contain the name of a record to be written. The format of the WRITE command is:

```
WRITE record-name
```

where the record-name is the name of a record on the ATTACHed dataframe.

Our example of dataframe C-80BYTE has only one record, C-80-BYTE-RECORD. If we look at the record as defined on SALF, we see only one record with a length of 80 characters.

NOTES

```

D B S INFORMATION EXPERT ----- LIST CONTENTS OF DATAFRAME C-80BYTE JSALFD
FIND DATA NAME:
ITEM / GROUP NAME          LVL  DISPL  TYPE  FLD  NBR  DEC  DT  OCCS
-----
C-80-BYTE-RECORD          00    0    A    80
C-BYTE                     01    0    A     1           80

ACTION: _____  1 Help  3 End  4 File Attr  6 Top  7 Pg Bwd  8 Pg Fwd
    
```

There is only one item defined under the record. This item is a recurring item. This item is 1 character, occurring 80 times.

There are two methods to move data into a record. The first method is to move data into the individual items defined in the record. The second method is to use alphanumeric sub stringing, specifying the beginning location of the record and the number of characters for each field. These methods can be combined.

Let's look at an example. We have a request to download Fixed Asset information. The user would like the following data:

Level 1	03 characters	FA-LEVEL-1
Level 2	03 characters	FA-LEVEL-2
Asset number	10 characters	FA-ASSET-NUMBER
Asset Description	35 characters	FA-ASSET-DESCRIPTION
Location	10 characters	FA-TAX-STATE
		FA-TAX-COUNTY
		FA-TAX-CITY
		FA-TAX-DISTRICT

There are 61 characters of information in this download request. The dataframe C-80BYTE is large enough to handle this request. The best approach to move data into this dataframe would be using the alphanumeric sub string method. When using this method, sub string each field into a record on the output dataframe. In our example, we are using the C-80BYTE dataframe that contains one record, C-80-BYTE-RECORD. To move each field using the sub string function, give a beginning location in the record and the number of characters to move. The following code could be used to move the data to the record:

NOTES

```

INPUT FAOASSET
RUN-TIME SELECT FA-LEVEL-1
WK-LOCATION(10A)           = FA-TAX-STATE . ;
                           FA-TAX-COUNTY . ;
                           FA-TAX-CITY . ;
                           FA-TAX-DISTRICT

ATTACH C-80BYTE CREATE
C-80-BYTE-RECORD(01 03)  = FA-LEVEL-1
C-80-BYTE-RECORD(04 03)  = FA-LEVEL-2
C-80-BYTE-RECORD(07 10)  = FA-ASSET-NUMBER
C-80-BYTE-RECORD(17 35)  = FA-ASSET-DESCRIPTION
C-80-BYTE-RECORD(52 10)  = WK-LOCATION
WRITE C-80-BYTE-RECORD
    
```

When the WRITE command is issued, any data moved into the record at that time is written to the output file.

Moving Numeric Fields

When moving data into an output record using the sub string approach, special procedures may be necessary when moving numeric data. When moving a numeric field into an alphanumeric field, no data conversion is performed. Let's look at what this means.

There are four types of numeric fields:

- N Display Numeric - Sign
- O Display Numeric – No Sign
- P Packed Numeric – Sign
- Q Packed Numeric – No Sign

When moving a numeric field to another numeric field, the data is converted to the characteristics of the new field type. In IE03, there was a discussion on how the different numeric fields are stored. When moving a numeric type to an alphanumeric type, data is moved exactly as it is stored in the numeric type. To review, this is how the number -123.45 would display with the different numeric types when moved to an alphanumeric field:

N	-123.45	1234N		
O	-123.45	12345		
P	-123.45)	(3 Positions)	13544 24d00
Q	-123.45	^	(3 Positions)	13544 24f00

As you can see, you should not move either of the packed fields to an alphanumeric field. The information is not decipherable. The data in the N type field is correct except for the last character, which carries a sign. The only remaining field would be the O type.

Before moving any numeric field to an alphanumeric field, first move the numeric field to an O type field, then move the O type field to the alphanumeric field.

When any numeric field is moved to an O type field, the sign is dropped. Normally, you will want to include the correct sign of a numeric field in the output file. To do this, create a separate field to contain the sign. Let's use this procedure to add the fixed asset cost to our example. This code is positioned before the WRITE command:

NOTES

```
WORK WK-SIGN (1A)
WORK WK-AMOUNT-O(10O2)
WORK WK-AMOUNT-A(10A)
WK-AMOUNT-O      =      FA-COST/1
WK-AMOUNT-A      =      WK-AMOUNT-O
IF FA-COST/1 LT 0
    WK-SIGN = '-'
ELSE
    WK-SIGN = '0'
END
C-80-BYTE-RECORD(62 01) = WK-SIGN
C-80-BYTE-RECORD(63 10) = WK-AMOUNT-A
```

The following are the first few records from the output dataset:

```
01400100040010007703 SOFA CULP-TANGIERS PACIF      DPI-OFFICE00000125559
0140010004001100TYPEWRITER SWINTEC 8016          C70-BUSH 00000055900
```

The above file is considered a fixed format file. The user needs to know where each field begins and ends to download this into Excel, Access, or any other PC package. Some users request the download file have delimiters placed between fields. Any special character can be used, as long as that character is not used within any field. The semi-colon is generally used to separate fields and is a default for Excel and Access. To use the semi-colon as a delimiter in an output file, you have to use the hexadecimal equivalent; otherwise Information Expert will think it's a continuation character. Let's add semi-colons to our download file example:

NOTES

```

INPUT FAOASSET
RUN-TIME SELECT FA-LEVEL-1
WK-LOCATION(10A)           =    FA-TAX-STATE . ;
                           =    FA-TAX-COUNTY . ;
                           =    FA-TAX-CITY . ;
                           =    FA-TAX-DISTRICT

ATTACH C-80BYTE CREATE
C-80-BYTE-RECORD(01 03)  =    FA-LEVEL-1
C-80-BYTE-RECORD(04 01)  =    X'5E'
C-80-BYTE-RECORD(05 03)  =    FA-LEVEL-2
C-80-BYTE-RECORD(08 01)  =    X'5E'
C-80-BYTE-RECORD(09 10)  =    FA-ASSET-NUMBER
C-80-BYTE-RECORD(19 01)  =    X'5E'
C-80-BYTE-RECORD(20 35)  =    FA-ASSET-DESCRIPTION
C-80-BYTE-RECORD(55 01)  =    X'5E'
C-80-BYTE-RECORD(56 10)  =    WK-LOCATION
C-80-BYTE-RECORD(66 01)  =    X'5E'
WORK WK-SIGN (1A)
WORK WK-AMOUNT-O(10O2)
WORK WK-AMOUNT-A(10A)
WK-AMOUNT-O      =    FA-COST/1
WK-AMOUNT-A      =    WK-AMOUNT-O
IF FA-COST/1 LT 0
    WK-SIGN = '-'
ELSE
    WK-SIGN = '0'
END
C-80-BYTE-RECORD(67 01) = WK-SIGN
C-80-BYTE-RECORD(68 10) = WK-AMOUNT-A
WRITE C-80-BYTE-RECORD
    
```

Since we are adding a delimiter between each field, our beginning locations have changed for each field. A sample of records written to the download file now looks like this:

```

014;001;0004001000;7703 SOFA CULP-TANGIERS PACIF ;DPI-OFFICE;00000125559
014;001;0004001100;TYPEWRITER SWINTEC 8016 ;C70-BUSH ;00000055900
    
```

INITIALIZE Command

The INITIALIZE command can be used to establish initial values for records, items or work fields. If the item is numeric, the item is initialized to zero. If the item is alphanumeric, the item is set to blank. The format for the INITIALIZE command is:

```

INITIALIZE item-name1 item-name2...
INITIALIZE record-name1 record-name2...
    
```

The item-name option is the item name from the dataframe or a work item whose value is to be initialized. The second version of the INITIALIZE command can reset a complete record.

It is a good practice to always initialize your output records before moving data to be written. This ensures the clearing of previous values that have been moved into your output record.

EXERCISE

NOTES

SCENARIO

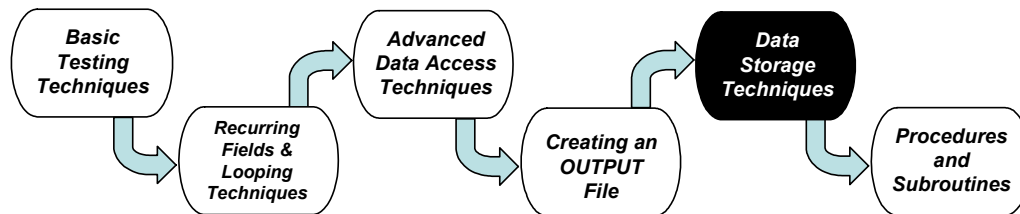
In this exercise you will learn to use the WRITE command to create an output dataset. You have been given a report request to create a download file for fixed assets.

1. In Source Management, go to *SMCR* and create a new report series C-IE04-WRITE-EXERCISE.
2. Use FAOASSET as the input dataframe. Use C-80BYTE as the output dataframe. Refer to the previous pages for detail information about the C-80BYTE dataframe. Refer to the previous pages for the assignment for the work field WK-LOCATION. Write the output file in the following order. Place a '%' as a delimiter between each field. Use the List command to also generate a report.

<u>Items</u>	<u>LTD</u>
a. FA-ASSET-NUMBER	10A
b. FA-COST/1	10N2
c. FA-ACQ-DATE	6A
d. WK-LOCATION	10A
e. FA-ADDITIONAL-DESCRIPTION-1	35A

3. CHECK the report series for errors. SAVE the report series when all errors have been corrected.
4. Submit the report series to prepare. Go to *Report Viewing (RVLS)* to insure the preparation was successful.
5. Go to *Job Submission (JSRU)* to run the report series.
6. Go to *Report Viewing (RVLS)* to view your results.

Data Storage Techniques



Overview

This section explains some data storage techniques. If a report request requires printing detail based on a total or average derived from the detail, data storage is necessary. For example, a report request may require you to print the detail invoices for a vendor if the average invoice amount is greater than a \$5,000.00. To determine the average, you would need to read, count and total all invoices for each vendor. Once all invoices are read for a vendor, you would need to compute the average. Then the average can be evaluated to determine if it meets your criteria. If the criteria were met, you would need to go back and retrieve the invoices already read to now include them on the report. This is where data storage can assist in completing this type of reporting task.

DATASTORE Command

The DATASTORE command can be used to create an area to store records that can be recalled at a later time. The DATASTORE command can display in the common section or in a report request. The format of the command is:

```
DATASTORE name CONTAINS item1 item2..... itemn
```

The option NAME is the name of the storage area. This option is from 1 to 8 characters and must begin with a letter. The name can contain letters, hyphens and numbers, but no spaces.

After the key word CONTAINS, specify the items you want to store. These items can be from a dataframe or a work field.

Let's look at an example. We receive a reporting task to print the paying entity, vendor number, invoice number, invoice date and payment amount for vendors with an average invoice amount of \$5,000.00. The following code would set up the datastore area:

NOTES

```
INPUT APINVC01
RUN-TIME SELECT AP-PAYING-ENTITY
SELECT AP-INVC-STATUS-CODE '12'
REPORT R1
DATASTORE AVGVEND CONTAINS;
    AP-PAYING-ENTITY;
    AP-VENDOR-NUMBER;
    AP-VENDOR-GROUP-NUMBER;
    AP-INVC-NUMBER;
    AP-INVC-DATE;
    AP-SCHED-PYMT-PAID-AMOUNT
```

STORE Command

The STORE command is used in conjunction with the DATASTORE command. The DATASTORE command sets up the storage area. The STORE command actually stores the values of the items listed in the DATASTORE. The item's values are stored when the STORE command is issued. The format of the STORE command is:

```
STORE datastore
```

where datastore is the name of a data storage area. The DATASTORE command must occur before the STORE command is issued.

Here is what our example would look like with the STORE command added and additional logic to calculate the average invoice amount.

```
INPUT APINVC01
RUN-TIME SELECT AP-PAYING-ENTITY
SELECT AP-INVC-STATUS-CODE '12'
REPORT R1
DATASTORE AVGVEND CONTAINS;
    AP-PAYING-ENTITY;
    AP-VENDOR-NUMBER;
    AP-VENDOR-GROUP-NUMBER;
    AP-INVC-NUMBER;
    AP-INVC-DATE;
    AP-SCHED-PYMT-PAID-AMOUNT
WHEN CHANGE OCCURS IN AP-VENDOR-GROUP-NUMBER
    WK-COUNT (70) = 0
    WK-TOTAL-PAYMENT(15P2) = 0
END
WHEN CHANGE SENSED IN AP-INVC-DATE
    STORE AVGVEND
    WK-COUNT = WK-COUNT + 1
    WK-TOTAL-PAMENT = WK-TOTAL-PAYMENT + AP-SCHED-PYMT-PAID-AMOUNT
END
WHEN CHANGE SENSED IN AP-VENDOR-GROUP-NUMBER
    WK-AVERAGE(15P2) = WK-TOTAL-PAYMENT / WK-COUNT
    IF WK-AVERAGE GE 5000.00
        ???
    END
END
```

RECALL Command

NOTES

The RECALL command can be used to retrieve the records that were placed in the data storage area. The data is recalled in the same order they are placed in the storage area. There are two forms of the RECALL command:

```
RECALL datastore
RECALL datastore UNTIL END
```

The first form of the RECALL command recalls one record. The second form of the command recalls all the information placed in data storage. The RECALL UNTIL EMPTY command must be terminated by a standalone END command.

Let's add the RECALL command to the above report series:

```
INPUT APINVC01
RUN-TIME SELECT AP-PAYING-ENTITY
SELECT AP-INVC-STATUS-CODE '12'
REPORT R1
DATASTORE AVGVEND CONTAINS;
  AP-PAYING-ENTITY;
  AP-VENDOR-NUMBER;
  AP-VENDOR-GROUP-NUMBER;
  AP-INVC-NUMBER;
  AP-INVC-DATE;
  AP-SCHED-PYMT-PAID-AMOUNT
WHEN CHANGE OCCURS IN AP-VENDOR-GROUP-NUMBER
  WK-COUNT (70) = 0
  WK-TOTAL-PAYMENT(15P2) = 0
END
WHEN CHANGE SENSED IN AP-INVC-DATE
  STORE AVGVEND
  WK-COUNT = WK-COUNT + 1
  WK-TOTAL-PAYMENT = WK-TOTAL-PAYMENT + AP-SCHED-PYMT-PAID-AMOUNT
END
WHEN CHANGE SENSED IN AP-VENDOR-GROUP-NUMBER
  WK-AVERAGE(15P2) = WK-TOTAL-PAYMENT / WK-COUNT
  IF WK-AVERAGE GE 5000.00
    RECALL AVGVEND UNTIL EMPTY
      LIST          AP-PAYING-ENTITY;
                   AP-VENDOR-NUMBER;
                   AP-VENDOR-GROUP-NUMBER;
                   AP-INVC-NUMBER;
                   AP-INVC-DATE;
                   AP-SCHED-PYMT-PAID-AMOUNT
    END
  PRINT AT 1 'AVERAGE INVOICE AMT : ' WK-AVERAGE
END
END
```

NOTES

EMPTY Command

Information stays in the data storage area until you recall it. If you determine the information you stored does not meet your criteria, you can “empty” the storage area. This command initializes the storage area. The format of the command is:

```
EMPTY datastore
```

If we add the EMPTY command to the above code, this is what the report series would look like:

```

WHEN CHANGE SENSED IN AP-VENDOR-GROUP-NUMBER
  WK-AVERAGE(15P2) = WK-TOTAL-PAYMENT / WK-COUNT
  IF WK-AVERAGE GE 5000.00
    RECALL AVGVEND UNTIL EMPTY
      LIST
        AP-PAYING-ENTITY;
        AP-VENDOR-NUMBER;
        AP-VENDOR-GROUP-NUMBER;
        AP-INVC-NUMBER;
        AP-INVC-DATE;
        AP-PYMT-AMOUNT
        AP-SCHED-PYMT-PAID-AMOUNT
      END
    PRINT AT 1 'AVERAGE INVOICE AMT : ' WK-AVERAGE
  ELSE
    EMPTY AVGVEND
  END
END

```

Here are the results:

14PT	541000588	03	1003405-002	01/06/04	4,660.00
14PT	541000588	03	1003406-002	01/06/04	4,470.00
14PT	541000588	03	1019586-002	09/01/05	4,470.00
14PT	541000588	03	1026280-001	04/29/05	18,800.00
14PT	541000588	03	10110530-002	01/06/04	4,470.00
AVERAGE INVOICE AMT:			7,374.00		
14PT	541000588	04	1019586-001	08/05/04	17,880.00
AVERAGE INVOICE AMT:			17,880.00		

ACTIVITY

SCENARIO

In this scenario you will use the DATASTORE, RECALL and EMPTY commands. A report has been requested to list all vendors that are one-time vendors. A one-time vendor is a vendor that has only one paid invoice. If the vendor is a one-time vendor, list the vendor number, vendor group number, vendor name, invoice number and invoice date. Do not print any detail invoice data.

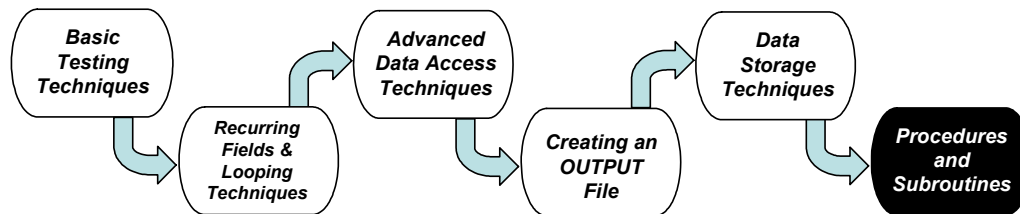
NOTES

1. In Source Management, go to *SMCR* and create a new report series C-IE04-DATASTORE-EXERCISE.
2. Use APINVC01 dataframe as your input dataframe. Select only paid invoices that have an AP-INVC-STATUS-CODE of '12'.
3. Use the following RELATE command to obtain the vendor name:

```
RELATE CVEND01 BY AP-VENDOR-GROUP-NUMBER  WHERE;  
      AP-PAYING-ENTITY = CV-PAYING-ENTITY AND;  
      AP-VENDOR-NUMBER =CV-VENDOR-NUMBER AND;  
      AP-VENDOR-GROUP-NUMBER = CV-VENDOR-GROUP-NUMBER  
IF CV-VENDOR-SETUP-RECORD.PRESENT  
      WK-VENDOR-NAME = CV-VENDOR-NAME  
ELSE  
      WK-VENDOR-NAME = 'VENDOR NOT FOUND'  
END
```

4. The report should list the Vendor number, Vendor group number, Vendor name, invoice number and invoice date.
5. CHECK the report series and correct any errors. Save the report series once all the errors have been corrected. Submit the report series to prepare.
6. In Report Viewing, go to *RVLS* to insure the preparation was successful.
7. In Job Submission, *JSRU*, run the report series.
8. In Report Viewing, *RVLS*, verify your results. There should be 6 one-time vendors listed.

Procedures and Subroutines



Overview

This section explains the Expert Language commands used to create procedures and subroutines. When writing Information Expert programs, there are times when common code is called for repeatedly. Instead of writing this common code multiple times, a procedure or subroutine can be established. Procedures and subroutines are Information Expert routines that can be executed multiple times.

PROCEDURE Command

A procedure is a set of Information Expert commands that is defined and used within the same section of a report series. Information Expert commands can be put in a procedure to make the report series more streamlined. A procedure is defined by a PROCEDURE command. The format of the PROCEDURE command is:

```
PROCEDURE proc-name
```

where proc-name is the name of the procedure. A procedure name can be from 1 to 30 characters, must begin with a letter, and can contain letters and hyphens, but no spaces. After the PROCEDURE command, add the Information Expert commands to execute. When all the commands within the procedure are coded, a standalone END command is used to terminate the procedure. Procedures are usually placed at the end of the section of the report series where they will be referenced. The code within the procedure will not be executed until the procedure is performed.

PERFORM Command

To execute a procedure, use the PERFORM command. The format of the PERFORM command is:

```
PERFORM proc-name
```

Let's look at an example. A report request has been delivered to list General Ledger balances for a given period. The report series must be able to handle current year and prior year. The following code could be used:

NOTES

```
INPUT GLOPENYR
RUN-TIME SELECT GL-COMPANY-ID
EXCLUDE GL-CENTER-ID '999999999999'
WORK PERIOD(20)
WORK CURRENT-FISCAL-YEAR(1A)
WORK WK-BALANCE(15P2)
VARIABLES ARE PERIOD;
                CURRENT-FISCAL-YEAR
IF CURRENT-FISCAL-YEAR = 'Y'
    WK-BALANCE = GL-PERIOD-ENDING-BALANCE/PERIOD - ;
    GLOPENYR:GL-PYR-PERIOD-ENDING-BALANCE/13
ELSE
    PERFORM CHECK-PRIOR-YEAR
END
IF WK-BALANCE NE 0
    EXTRACT DATA FOR R1
END
NEXT RECORD
PROCEDURE CHECK-PRIOR-YEAR
    IF GL-PRIOR-YEAR-OPEN-INDICATOR = '0'
        WK-RELATIVE-YEAR = '01'
        RELATE GLCLOSyr WHERE;
            WK-RELATIVE-YEAR = GL-RELATIVE-YEAR
        WK-BALANCE = GLCLOSyr:GL-PYR-PERIOD-ENDING-BALANCE
    ELSE
        WK-BALANCE = GLOPENYR:GL-PYR-PERIOD-ENDING-BALANCE/PERIOD
    END
END

REPORT R1
LIST GL-COMPANY-ID;
    GL-ACCOUNT-ID;
    GL-CENTER-ID;
    WK-BALANCE HEADING IS 'BALANCE'
```

SUBROUTINE Command

A subroutine is similar to a procedure in that it is Information Expert commands that can be performed when requested. Subroutines must be prepared. When creating a subroutine online using *Source Management Create (SMCR)*, the type must be 3. Subroutine names use the same characteristics as report series names: from 1-30 characters, beginning with a letter and possibly containing letters, numbers and hyphens, and no spaces.

NOTES

```
D B S INFORMATION EXPERT ----- DEFINE TYPE OF SOURCE MEMBER SMCR

ENTER MEMBER TYPE BELOW:

  1 - REPORT SERIES          5 - RUN STATEMENTS
  2 - PRINT-SERIES          6 - DICTIONARY MAINTENANCE
  3 - SUBROUTINES           7 - UTILITY STATEMENTS
  4 - MISCELLANEOUS

MEMBER TYPE   ===>  _
MEMBER NAME   ===> _____
LIBRARY NAME  ===> MSAUSER

ACTION: _____

PRESS:      ENTER Process   PF1 Help   PF3 Cancel Create
```

Subroutines are standalone Information Expert Code. Subroutines can be executed by any report series. A subroutine would be used if there were commands that must be executed by multiple report series. If you placed the code in each report series, when that code changes, you would have to change and re-prepare each report series. If you place the code in a subroutine, you would only have to change the subroutine and re-prepare the subroutine.

The first command in a subroutine is the PARMs ARE command. This command lists the parameters that are passed to and from the main program and the subroutine. The format of the PARMs ARE command is:

```
PARMS ARE parm1 parm2 parm3 parm4
```

The parameters are located after the keyword ARE. Subroutines can only pass four parameters. Each parameter can be an item, group or a record. Obviously, even though there are only four parameters, it does not limit the number of data fields that can be passed.

Parameters are positional. The first parameter on the PARM command corresponds to the first parameter when the subroutine is executed in the report series. Each corresponding parameter must be the EXACT same size. The parameter does not have to be the same type, just the same size. If the parameter is an item, group or record that resides on the data dictionary, a LTD does not have to be specified. If the parameter does not reside on the data dictionary, a Length, Type and Decimal point or Date format (LTD) must be specified.

There are commands that cannot be used in a subroutine. No input / output commands can be used. This includes INPUT, MATCH, READ, READ NEXT, RELATE. No sequence break processing commands can occur, like WHEN CHANGE SENSED, WHEN CHANGE OCCURS, FIRST TIME DO, LAST TIME DO. Printing commands cannot occur in a subroutine. That excludes commands like PRINT, LIST, DEFINE PAGEHEADINGS/

NOTES

PAGEFOOTINGS from being used in a subroutine. Refer to the Expert Language Reference Guide for a complete list of statements that are not permitted in a subroutine. Subroutines are generally used for calculations, routines and assignment.

Once a subroutine has been coded, it must be prepared. If the subroutine is being prepared from the Online Support Facility, you can prepare a subroutine by submitting the member from *Source Management List Member screen (SMLS)*, while editing the member. The member can also be submitted using the *Job Submission Prepare Series/Subroutines screen (JSPR)*. Once the member has been submitted to be prepared, check *Report Viewing List Series (RVLS)* to insure the preparation was successful.

PERFORM SUBROUTINE Command

To execute a subroutine, use the PERFORM SUBROUTINE command. The format of this command is:

```
PERFORM SUBROUTINE sub-name USING parm1 parm2 parm3 parm4
```

The sub-name is the name of a subroutine that has been successfully prepared. The parameters that are being passed to and from the subroutine display after the keyword USING. These parameters must correspond to the parameters on the PARMS command in the subroutine. Data is passed by position (or address) by length between the report series and subroutine. These parameters must be the exact same LENGTH or results may be unpredictable.

Let's look at an example that is used in Fixed Assets. When an asset is added to Fixed Assets, the expense object is entered into FA-LEVEL-14-VALUE. For reporting purposes, the expense object must be converted to the corresponding asset account. A subroutine was created to perform this task. All fixed asset reports that require the expense object conversion to the asset account perform this same subroutine. The following is an excerpt from one of the Fixed Asset programs that calls this subroutine:

```
WORK WK-ASSET-ACCT      (4A)  
WORK WK-OBJECT          (4A)  
  
WK-OBJECT = FA-LEVEL-14-VALUE(1 4)  
PERFORM SUBROUTINE C-CONVERT-TO-ASSET-ACCOUNT USING;  
                    WK-OBJECT WK-ASSET-ACCT
```

There are two parameters being used in this subroutine, both are work fields. Let's look at the code for the subroutine:

PARMS ARE WK-EXP-OBJECT(4A) WK-ASSET-ACCOUNT(4A)

NOTES

* CONVERT EXPENDITURE OBJECT TO ASSET OBJECT

```
WK-ASSET-ACCOUNT = '7999'  
IF WK-EXP-OBJECT GE '7000' AND WK-EXP-OBJECT LE '7999'  
  WK-ASSET-ACCOUNT = WK-EXP-OBJECT  
  EXIT SUBROUTINE  
END
```

```
IF WK-EXP-OBJECT GE '4100' AND WK-EXP-OBJECT LE '4199'  
  WK-ASSET-ACCOUNT = '7000'  
  EXIT SUBROUTINE  
END
```

```
IF WK-EXP-OBJECT GE '4200' AND WK-EXP-OBJECT LE '4299'  
  WK-ASSET-ACCOUNT = '7100'  
  EXIT SUBROUTINE  
END
```

```
IF WK-EXP-OBJECT GE '4300' AND WK-EXP-OBJECT LE '4399'  
  WK-ASSET-ACCOUNT = '7100'  
  EXIT SUBROUTINE  
END
```

If we isolate the parameters from the report series and the subroutine we see:

<u>Report Series</u>	<u>Subroutine</u>
WK-OBJECT (4A)	WK-EXP-OBJECT (4A)
WK-ASSET-ACCT(4A)	WK-ASSET-ACCOUNT(4A)

The parameter names do not have to be the same in the report series and subroutine. The length MUST be the same or results may be unpredictable. In the above example there are two parameters. The first parameter in the report series, WK-OBJECT passes its values to the subroutine as WK-EXP-OBJECT. In the subroutine, WK-EXP-OBJECT is checked in ranges to determine the correct value for WK-ASSET-ACCOUNT. The value that is set for WK-ASSET-ACCOUNT is then passed to the report series as WK-ASSET-ACCT. The report series then uses WK-ASSET-ACCT in the report request to print the asset account.

Another example of a subroutine can be found in some of the General Ledger programs. Subroutine AA-ACCOUNT-LOOKUP is used to identify agency accounts. The following is code from the AA-ACCOUNT-LOOKUP subroutine:

NOTES

PARMS ARE WK-BASEACCT (9A) WK-AA-ACCT-SW (1A)

```
WK-AA-ACCT-SW = 'N'  
IF (WK-BASEACCT GE '5361 ' AND WK-BASEACCT LT '536799999')  
  WK-AA-ACCT-SW = 'Y'  
  EXIT SUBROUTINE  
END  
IF (WK-BASEACCT GE '5369 ' AND WK-BASEACCT LT '536999999')  
  WK-AA-ACCT-SW = 'Y'  
  EXIT SUBROUTINE  
END  
IF (WK-BASEACCT GE '5371 ' AND WK-BASEACCT LT '537199999')  
  WK-AA-ACCT-SW = 'Y'  
  EXIT SUBROUTINE  
END
```

There are two parameters in this subroutine: Parameter WK-BASEACCT and WK-AA-ACCT-SW. The following is code from a report series that calls the AA-ACCOUNT-LOOKUP subroutine and also uses a procedure:

```
WORK WK-BASEACCT    (9A)  
WORK WK-AA-ACCT-SW  (1A)  
WK-BASEACCT = WK-ACCOUNT-ID(1 9)  
WK-SUB-ACCOUNT = WK-ACCOUNT-ID(7 3)  
IF WK-SUB-ACCOUNT NE ' '  
  PERFORM SUBROUTINE AA-ACCOUNT-LOOKUP USING ;  
    WK-BASEACCT WK-AA-ACCT-SW  
  PERFORM MASTER-FILE-LOOKUP  
ELSE  
  PR-ACCOUNT-DESC = GL-ACCT-DESCRIPTION  
END  
< other code >  
  
PROCEDURE MASTER-FILE-LOOKUP  
  IF WK-AA-ACCT-SW = 'Y'  
    C-GL-COMPANY-ID = GL-COMPANY-ID  
    C-GL-ACCOUNT-ID = WK-BASEACCT  
    C-GL-CENTER-ID = GL-CENTER-ID  
    C-GL-CHARGE-TYPE = 'B'  
    C-GL-RECORD-TYPE = 'B'  
    RELATE C-GMP11  
    IF C-GL-ACCOUNT-INFO-RECORD.PRESENT  
      PR-ACCOUNT-DESC = C-GL-ACCOUNT-DESCRIPTION  
    ELSE  
      C-GL-ACCOUNT-ID = WK-BASEACCT(1 4) . 'AA'  
      C-GL-COMPANY-ID = 'AAAA'  
      RELATE C-GMP11  
      IF C-GL-ACCOUNT-INFO-RECORD.PRESENT  
        PR-ACCOUNT-DESC = C-GL-ACCOUNT-DESCRIPTION  
      ELSE  
        PR-ACCOUNT-DESC = 'NO DESC FOUND'  
    END  
  END  
  ELSE  
    <other code>  
  END  
END
```

Let's compare the parameters:

NOTES

<u>Subroutine</u>		<u>Report Series</u>	
WK-BASEACCT	9A	WK-BASEACCT	9A
WK-AA-ACCT-SW	1A	WK-AA-ACCT-SW	1A

We can see that the length for each item in the subroutine matches the length for each item in the report series. In this example, the value in WK-BASEACCT from the report series is passed to the item of the same name in the subroutine. Item WK-BASEACCT is evaluated to determine if it's an agency account. If it is, item WK-AA-ACCT-SW is set to 'Y'.

These subroutines also demonstrate the use of the EXIT command. Once a true condition is met, the WK-AA-ACCT-SW is set to 'Y', the next command is EXIT SUBROUTINE. When this EXIT command is executed, the control is passed back to the report series. The other commands in this subroutine are not executed. This EXIT command has increased the efficiency of the subroutine processing.

Converting Alphanumeric to Numeric

Numeric fields are the only fields that are allowed in a calculation. What if the item contains numeric data, but the item type is defined as alphanumeric? Even though the data in the field is numeric, it is not allowed in a calculation because of the item type. There are several instances of this that occur within the NCAS system.

In a previous section we learned to create an output file; we wrote a dataset using the C-80BYTE dataframe. This was our final report series:

NOTES

```

INPUT FAOASSET
RUN-TIME SELECT FA-LEVEL-1
ATTACH C-80BYTE CREATE
WK-LOCATION(10A)          =      FA-TAX-STATE . ;
                           FA-TAX-COUNTY . ;
                           FA-TAX-CITY . ;
                           FA-TAX-DISTRICT

C-80-BYTE-RECORD(01 03) =      FA-LEVEL-1
C-80-BYTE-RECORD(04 01) =      X'5E'
C-80-BYTE-RECORD(05 03) =      FA-LEVEL-2
C-80-BYTE-RECORD(08 01) =      X'5E'
C-80-BYTE-RECORD(09 10) =      FA-ASSET-NUMBER
C-80-BYTE-RECORD(19 01) =      X'5E'
C-80-BYTE-RECORD(20 35) =      FA-ASSET-DESCRIPTION
C-80-BYTE-RECORD(55 01) =      X'5E'
C-80-BYTE-RECORD(56 10) =      WK-LOCATION
C-80-BYTE-RECORD(66 01) =      X'5E'
WORK WK-SIGN (1A)
WORK WK-AMOUNT-O(10O2)
WORK WK-AMOUNT-A(10A)
WK-AMOUNT-O      =      FA-COST/1
WK-AMOUNT-A      =      WK-AMOUNT-O
IF FA-COST/1 LT 0
    WK-SIGN = '-'
ELSE
    WK-SIGN = '0'
END
C-80-BYTE-RECORD(67 01) = WK-SIGN
C-80-BYTE-RECORD(68 10) = WK-AMOUNT-A
WRITE C-80-BYTE-RECORD
    
```

We can now write a report series using the C-80BYTE dataframe as input.

```

INPUT C-80BYTE
WORK WK-ASSET-NUMBER(10A)
WORK WK-AMOUNT-A(10A)
WORK WK-SIGN(1A)

WK-ASSET-NUMBER =      C-80-BYTE-RECORD(09 10)
WK-SIGN          =      C-80-BYTE-RECORD(67 01)
WK-AMOUNT-A      =      C-80-BYTE-RECORD(68 10)
    
```

We have isolated the amount field, but it's in an alphanumeric work field. Remember, subroutines pass parameters by position. Let's look at the following subroutine that we will call C-CONVERT-ALPHA-TO-NUMERIC:

```

PARMS ARE WK-AMOUNT-I (10A) WK-AMOUNT-O(10A)
WK-AMOUNT-O = WK-AMOUNT-I
    
```

There are two parameters, both defined as 10 alphanumeric characters. Since they are both alphanumeric types, we can move WK-AMOUNT-I to WK-AMOUNT-O. Let's add the code in the report series to call the subroutine:

```
WORK WK-AMOUNT-N(10N2)
PERFORM SUBROUTINE C-CONVERT-ALPHA-TO-NUMERIC USING;
      WK-AMOUNT-A WK-AMOUNT-N
IF WK-SIGN = '-'
      WK-AMOUNT-N = WK-AMOUNT-N * -1
END
```

NOTES

The item WK-AMOUNT-A passes the alphanumeric field to the subroutine into parameter WK-AMOUNT-I. WK-AMOUNT-O passes the alphanumeric field back to the report series into a NUMERIC field WK-AMOUNT-N. Since the data is passed to and from report series and subroutine by address, this process is acceptable.

When using a subroutine to convert data from alphanumeric to numeric, it is up to the user to ensure the field being converted contains only numbers. If the field contains any special characters, blanks, or alpha characters, an abnormal termination will occur.

ACTIVITY

SCENARIO

In this scenario, you will write a subroutine that will convert an alphanumeric field to a numeric field. You will read the data file that was created in C-IE04-WRITE-EXERCISE. You have been asked to adjust the cost of fixed assets by 5% and give a total of the cost and a total of the adjusted cost.

1. In *Source Management*, go to *SMCR* and create a new subroutine member called C-IE04-CONVERT-TO-NUMERIC.
2. Edit the subroutine member. There will be two parameters. The lengths of the parameters are both 10. The subroutine will convert the alphanumeric field to a numeric field.
3. CHECK and SAVE the subroutine once all errors have been corrected.
4. Submit the subroutine for preparation.
5. Go to *Report Viewing*, *RVLS*, to ensure the subroutine was prepared successfully.
6. In *Source Management*, go to *SMCR* and create a new report series member called C-IE04-SUBROUTINE-EXERCISE.
7. The primary dataframe will be C-80BYTE. Parse out the asset number and asset cost.
8. Call the subroutine to convert the asset cost to a numeric cost field. Multiply the numeric cost field by 1.05 to give the adjusted cost.

NOTES

9. Total the cost and adjusted code by #REPORTID.
10. List asset number, asset cost and adjusted cost, giving a total at the end of the report.
11. CHECK your report series and SAVE after correcting any errors.
12. Submit the report series for preparation.
13. In *Report Viewing, RVLS*, check to ensure the report series was prepared successfully.
14. In *Job Submission, JSRU*, select the report series to run.
15. In *Report Viewing, RVLS*, review your results.

QRG 1: OUTPUT Dataframes

<i>Dataframe</i>	<i>LRECL</i>	<i>DDNAME</i>	<i>RECORD NAME</i>
C-80BYTE	080	C80BYTE	C-80-BYTE-RECORD
C-1000BYT	100	C100BYTE	C-100-BYTE-RECORD
C-1000RC	1000	C100REC	C-10000-BYTE-REC
C-110BYT	110	C110BYTE	C-110-BYTE-RECORD
C-127BYT	127	C127BYTE	C-127-BYTE-RECORD
C-154BYT	154	C154BYTE	C-154-BYTE-RECORD
C-158BYT	158	C158BYTE	C-158-BYTE-RECORD
C-158BY2	158	C158BYT2	C-158-BYT2-RECORD
C-1814RC	1814	C1814REC	C-1814-BYTE-RECORD
C-225DF	225	TXN225	C-225-BYTE-RECORD
C-250BYT	250	C250BYTE	C-250-BYTE-RECORD
C-300BYT	300	C300BYTE	C-300-BYTE-RECORD
C-350BYT	350	C350BYTE	C-350-BYTE-RECORD
C-540BYT	540	C540BYTE	C-540-BYTE-RECORD
C-600BYT	600	C600BYTE	C-600-BYTE-RECORD
C-80BYT1	80	C80BYT1	C-80-BYTE-RECORD-1
C-80BYT2	80	C80BYT2	C-80-BYTE-RECORD-2
C-80BYT3	80	C80BYT3	C-80-BYTE-RECORD-3
C-800REC	800	C800REC	C-800-BYTE-REC
C-814OUT	814	C814OUT	C-814-BYTE-OUT
C-814REC	814	C814REC	C-814-BYTE-REC
C-88BYTE	88	C88BYTE	C-88-BYTE-RECORD

