

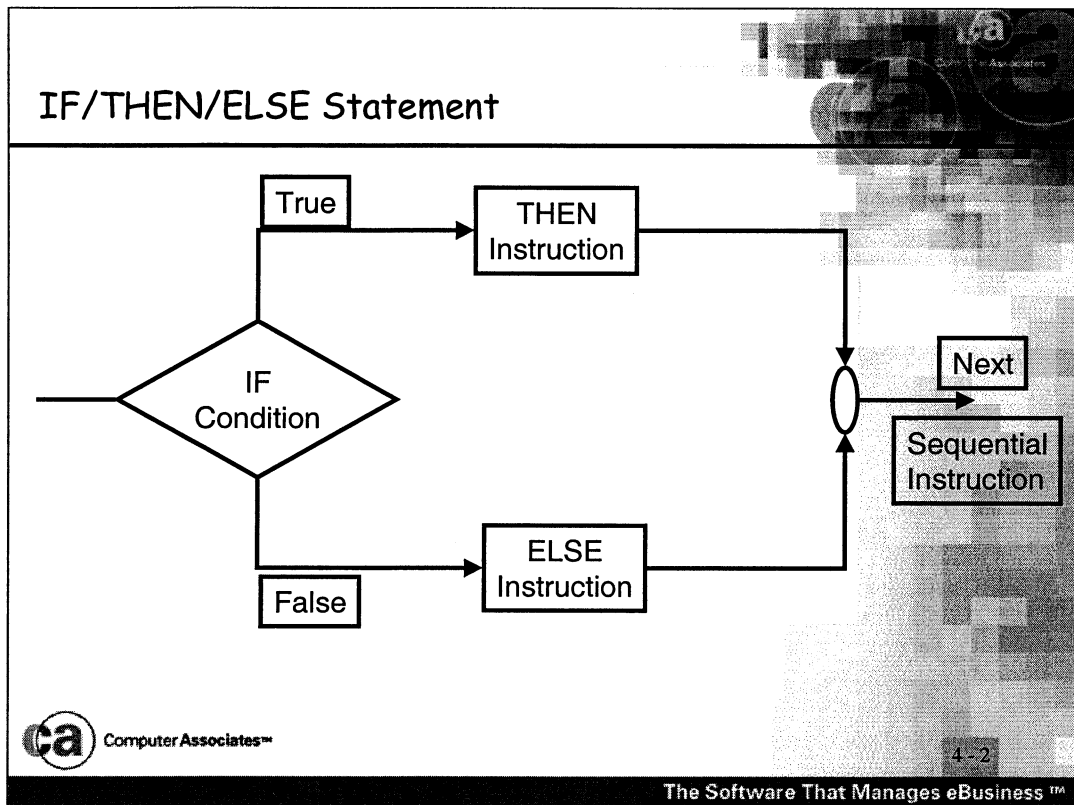
Flow Control Instructions

Section 4

ca Computer Associates™

The Software That Manages eBusiness™

4 - 1



IF/THEN/ELSE Statement

- Format
 - IF expression THEN instruction
 - [ELSE instruction]
- Expression must evaluate to 1 or 0 (true or false)
- Every IF must have THEN and an instruction indicating what to do if the expression is true
- The ELSE clause indicates what to do if the expression is false, optional.



IF/THEN/ELSE Statement

- Example

```
job_name = "PAYROLL"  
system = "UP"  
IF job_name = "PAYROLL" THEN  
    SAY "Load payroll cheques"  
IF system = "UP" THEN  
    SAY "System is up"  
ELSE  
    SAY "System is down"
```



Test Exercise 41

- Write a REXX program to test the statements below.
- Correct the code where necessary

```
test_value = 1
IF test_value = 1 SAY "Yes"
IF test_value = 1
  THEN SAY "Yes"
IF test_value = 1 THEN SAY "Yes" ELSE SAY "No"
IF test_value = 1 ELSE SAY "No"
IF test_value = 1
  THEN
    SAY "Yes"
  ELSE
    SAY "No"
```



Computer Associates™

4 - 5

The Software That Manages eBusiness™

Nested IF statements

- IF statements also may be nested within IF statements

```
PARSE ARG age
IF age < 65 THEN
  IF age > 21 THEN
    SAY "Over 21 and under 65"
  ELSE
    IF age >= 16 THEN
      SAY "Between 16 and 21"
    ELSE
      SAY "Under 16"
  ELSE
    SAY "65 or over"
```



Comparative Operators

- Compare two terms and return 1 if the result is true and 0 if then result is false
- Normal comparison
 - = equal
 - \= not equal (can also use not sign, X'5F')
 - > greater than
 - < less than
 - >< greater than or less than (same as not equal)
 - >= greater than or equal to
 - <= less than or equal to
 - \< not less than
 - \> no greater than



4 - 7

The Software That Manages eBusiness™

== strictly equal

\== not strictly equal (can also use not sign, X'5F')

Also in 3270 :

¬= not equal

Comparative Operators

- When REXX compares two non-numeric values, it ignores leading and trailing spaces
 - " REXX " = "REXX"
 - Would evaluate as true
- When REXX compares two numeric values it ignores leading and trailing zeros.
 - 00000000012 = 12
 - 12 = 12.000
 - Would evaluate to true



Comparative Operators

- Strictly means that the two values must match each other.
 - `00000000000012 == 12`
 - Would be false
 - `" REXX " == "REXX"`
 - Would evaluate as false



Logical Operators

- Logical operators combine two comparisons return 0 or 1.
- Types of logical operators.

&	AND
	OR
&&	EXCLUSIVE OR
\	NOT



4 - 10

The Software That Manages eBusiness™

& AND - returns a 1 (true) if both comparisons are true, and a 0 (false) otherwise - performs a logical AND operation

| OR - returns a 1 (true) if at least one comparison of several is true, and a 0 (false) otherwise - performs a logical or operation

&& EXCLUSIVE OR - returns a 1 (true) if ONLY one of a group of comparisons is true, and a 0 (false) otherwise - performs a logical exclusive OR function

\ NOT - returns the reverse logical value for an expression returns false if expression resolves to true, and true if the expression resolves to false

Multiple Logical Operators

- When multiple logical operators are used, &s are evaluated before |s.

```
test_value = 1
old_value = 2000
new_value = 3
IF test_value = 1 & (old_value = 2 | new_value = 3) THEN
  SAY "All ok"
```



THEN DO and ELSE DO

- This groups several statements together so that REXX will treat them as one instruction
- Often you need to execute more than one instruction in a THEN or ELSE clause

```
system_state = "UP"  
IF system_state = "UP" THEN DO  
    SAY "The system should be down"  
    system_state = "DOWN"  
END
```



SELECT statement

- Format
 - SELECT
 - WHEN
 - OTHERWISE
 - END

```
system_state = "UP"
SELECT
  WHEN system_state = "UP" THEN
    system_state = "DOWN"
  WHEN system_state = "DOWN" THEN
    system_state = "UP"
  WHEN system_state = "FAIL" THEN
    system_state = "DOWN"
  WHEN system_state = "WARNING" THEN
    system_state = "ERROR"
  OTHERWISE SAY "System state invalid"
END
```



Computer Associates™

4 - 13

The Software That Manages eBusiness™

Unlike IF with ELSE, the SELECT statement requires the OTHERWISE for all false conditions.

NOP

- Dummy instruction that has no effect
- Often used with and IF and SELECT

```
system_state = "UP"  
SELECT  
  WHEN system_state = "UP" THEN  
    system_state = "DOWN"  
  WHEN system_state = "DOWN" THEN  
    system_state = "UP"  
  WHEN system_state = "FAIL" THEN  
    system_state = "DOWN"  
  WHEN system_state = "WARNING" THEN  
    system_state = "ERROR"  
  OTHERWISE NOP  
END
```



Work Section 4.1

- Re-Write the "AGE" nested IF as four separate IF statements in order to perform the same function, in a REXX program. Assign the value AGE as an argument.

```
ex 'clcs.iulc00.rexx(rx10141)' '65'

65 or over
***
```

AGE =	Result
10	
21	
65	
70	
Your age	



Computer Associates™

4 - 15

The Software That Manages eBusiness™

```
IF age < 65 THEN
  IF age > 21 THEN
    SAY "Over 21 and under 65"
  ELSE
    IF age >= 16 THEN
      SAY "Between 16 and 21"
    ELSE
      SAY "Under 16"
ELSE
  SAY "65 or over"
```

Enter the results in the table above.
Hint do not use any else statements.

Work Section 4.2

- Re-Write Work section 4.1 "AGE" Using the select Statement

```
ex 'clcs.iulc00.rexx(rx10131)' '65'

65 or over
***
```

AGE =	Result
10	
21	
65	
70	
Your age	



Computer Associates™

4-16

The Software That Manages eBusiness™

```
IF age < 65 THEN
  IF age > 21 THEN
    SAY "Over 21 and under 65"
  ELSE
    IF age >= 16 THEN
      SAY "Between 16 and 21"
    ELSE
      SAY "Under 16"
  ELSE
    SAY "65 or over"
```

Enter the results in the table above.

Additional program

- Write a REXX program to display the tax paid for each of the codes below given entered at the screen:

Tax Code as a Percent	Result
10	
20	
50	
70	
80	

```
Please enter your salary.  
12000  
Please enter your TAX band.  
70  
Your tax for : 12000 : is : 8400.0  
***
```



Computer Associates™

4 - 17

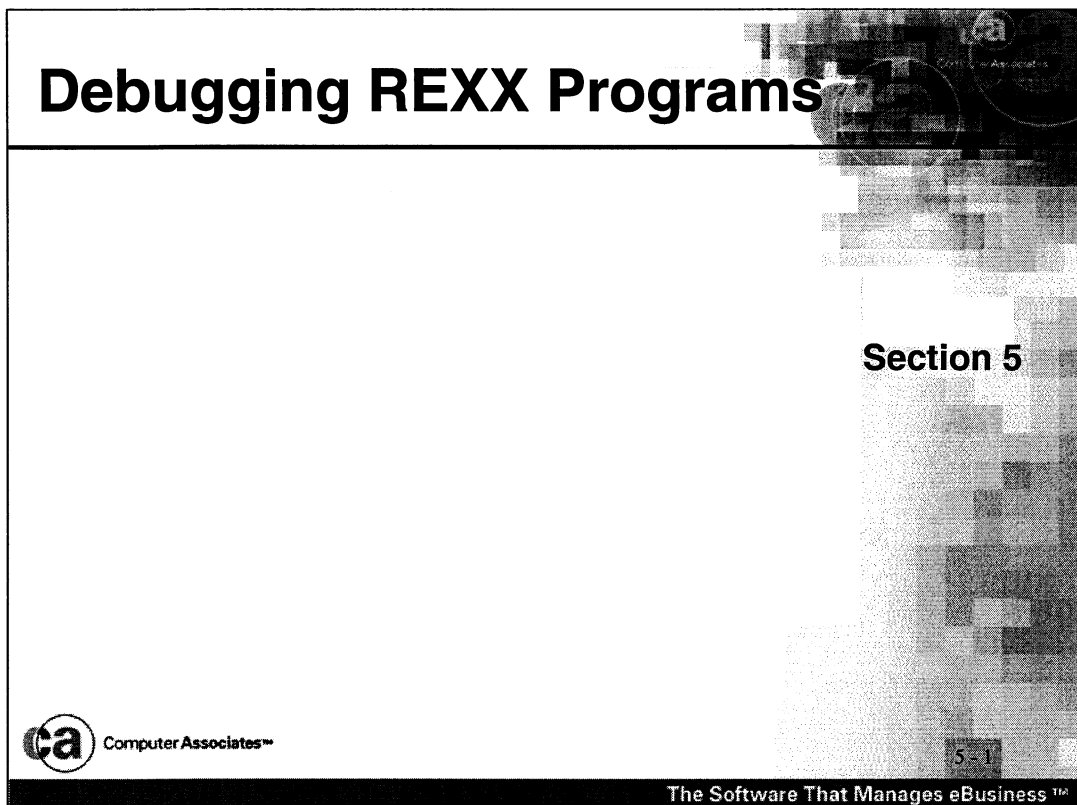
The Software That Manages eBusiness™

Section end

ca Computer Associates™

4 - 18

The Software That Manages eBusiness™



Debugging

- TRACE Instruction
 - Syntax errors
 - logic problems
 - often used when creating programs
- Error Trapping
 - abnormal situations
 - often used in production
- Dependent REXX platform
 - examples using TSO/E REXX



Interactive Tracing

- Before execution of a REXX program
 - EXECUTIL TS
- During execution of a REXX program
 - PA1 or ATTN
 - HI
- Commands in the REXX program
 - TRACE [?!] option
 - EXECUTIL TS|TE|HT|RT|HI



TRACE Instruction

- trace setting :
 - A
 - (All) - all REXX clauses are traced before they execute - this single option enables all of those below Call host commands are traced before execution.
 - C
 - (Commands) - all TSO/E commands are traced before they are executed and any non-zero return code returned by a TSO/E command is also traced; as an example, the 'TRACE !C' instruction causes TSO/E commands to be traced but not executed.
 - E
 - (Error) - any TSO/E command returning a non-zero return code is traced after it executes
 - I
 - (Intermediates) - all clauses are traced before they execute, and the 'intermediate' (working) results of evaluations of expressions and any substituted names are traced too



5 - 4

The Software That Manages eBusiness™

Format :

TRACE <trace setting>

TRACE Instruction

- L
 - (Labels) - labels on instructions that execute are traced
- N
 - (Normal) - a TSO/E command that returns a negative return code is traced after it executes - this trace option is the default if no other is specified
- O
 - (Off) - turns off ALL tracing
- R
 - (Results) - all REXX clauses are traced before they execute, and the final results of evaluating each expression are also traced. The values that are assigned via a PULL, ARG, and PARSE instructions are traced.
- S
 - (Scan) - all clauses in the exec are traced, but not executed. Checking for missing END statements is carried out. 'TRACE S' only works if the 'TRACE S' clause is not imbedded in any other instruction, such as an INTERPRET or an internal routine.



TRACE Example - A

```
TRACE A
"test"
old_name = "Fred"
SAY "Please enter name : "
PARSE PULL tst_name
PARSE VAR tst_name new_name .
IF new_name = old_name THEN DO
    SAY "Hello Fred"
END
ELSE DO
    NOP
END
```



TRACE Example - A

```
12 *-* "test"  
    >>> "test"  
MISSING DSNAME  
13 *-* old_name = "Fred"  
14 *-* SAY "Please enter name : "  
Please enter name :  
15 *-* PARSE PULL tst_name  
Fred  
16 *-* PARSE VAR tst_name new_name .  
17 *-* IF new_name = old_name  
    *-* THEN  
    *-* DO  
18 *-*   SAY "Hello Fred"  
Hello Fred  
19 *-* END  
***
```

```
12 *-* : Source Code  
>>>   : Result of statement  
MISSING DSNAME : TSO Command
```



Computer Associ

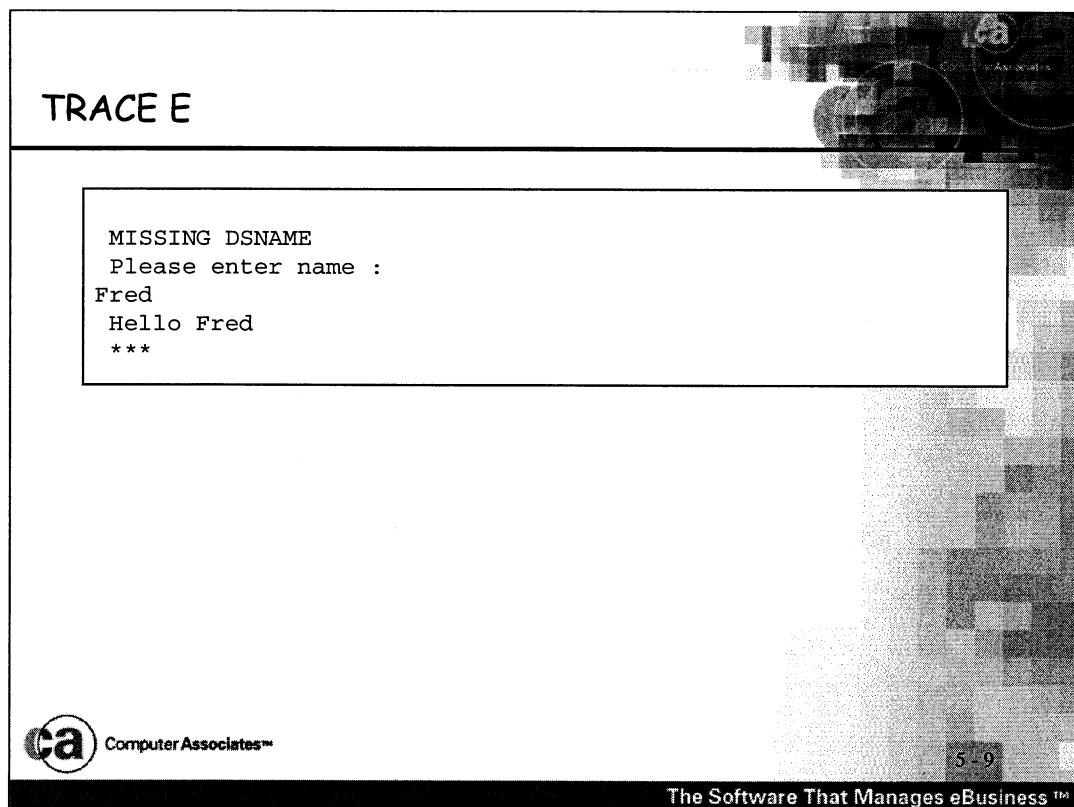
The Software That Manages eBusiness™

TRACE C

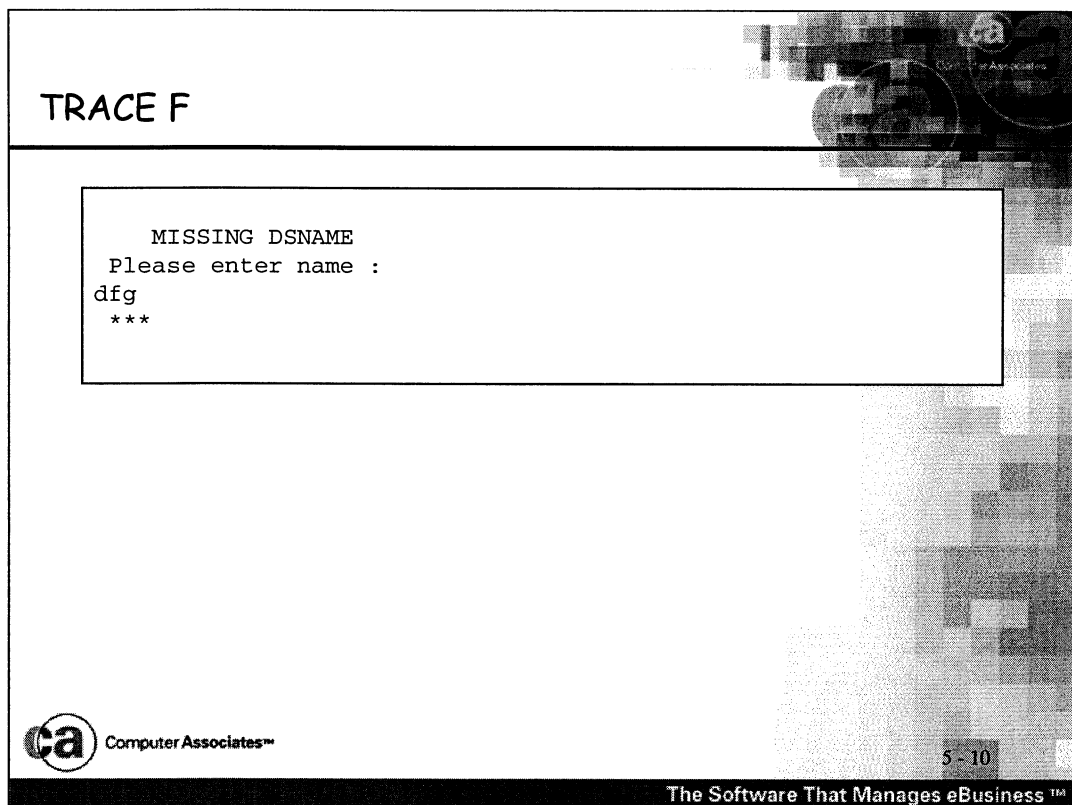
```
12 *-* "test"  
    >>> "test"  
MISSING DSNAME  
Please enter name :  
Fred  
Hello Fred  
***
```



The Software That Manages eBusiness™



Traces error after execution.



F - Failure (Same as Normal)

TRACE I

```
12 *-* "test"
    >L> "test"
MISSING DSNAME
13 *-* old_name = "Fred"
    >L> "Fred"
14 *-* SAY "Please enter name : "
    >L> "Please enter name : "
Please enter name :
15 *-* PARSE PULL tst_name
bob
    >>> "                                bob"
16 *-* PARSE VAR tst_name new_name .
    >>> "bob"
    >.> ""
17 *-* IF new_name = old_name
    >V> "bob"
    >V> "Fred"
    >O> "0"
20 *-* ELSE
    *-* DO
21 *-* NOP
22 *-* END
```



TRACE I

>L> : Literal String

>.> : this line shows the value assigned to a placeholder in a parse template during parsing.

>V> : this line shows the contents of a variable

>O> : this line shows the result of an operation on two terms



5 - 12

The Software That Manages eBusiness™

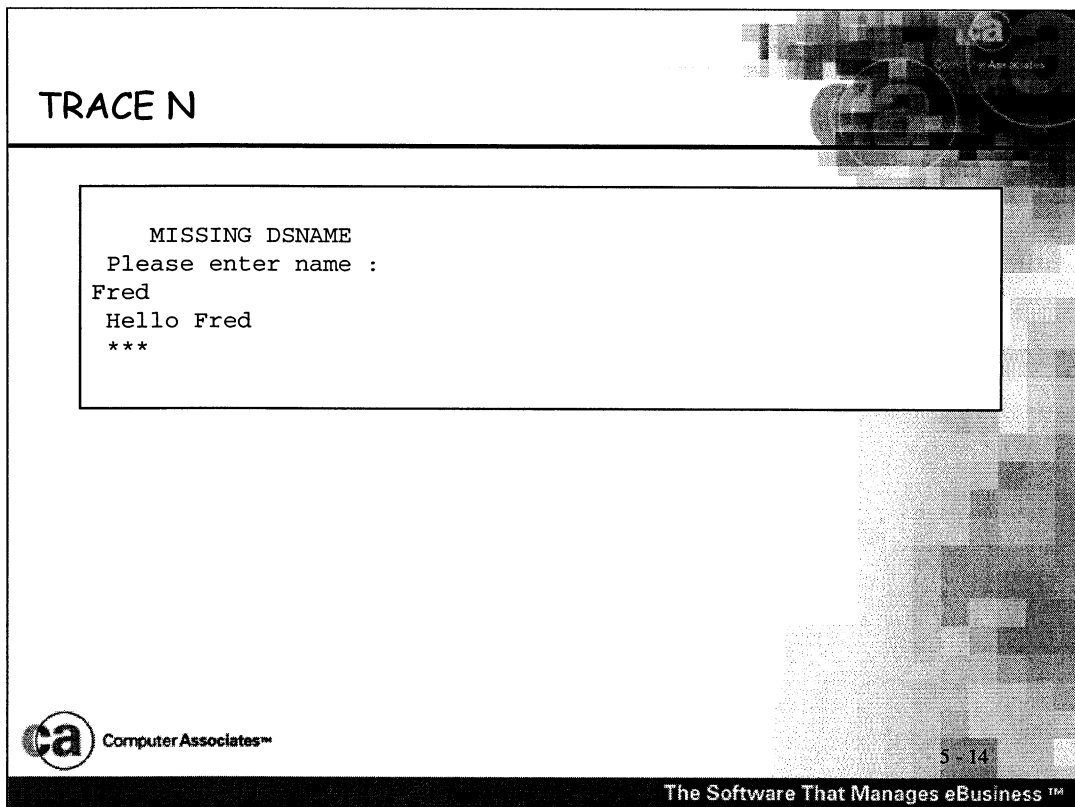
TRACE L

```
MISSING DSNAME  
Please enter name :  
Fred  
Hello Fred  
***
```



5-13

The Software That Manages eBusiness™



Normal - Default (Same as Failure)

TRACE O

```
MISSING DSNAME  
Please enter name :  
bob  
***
```



5 - 15

The Software That Manages eBusiness™

TRACE R

```
12 *-* "test"
    >>> "test"
MISSING DSNAME
13 *-* old_name = "Fred"
    >>> "Fred"
14 *-* SAY "Please enter name : "
    >>> "Please enter name : "
Please enter name :
15 *-* PARSE PULL tst_name
bob
    >>> "                                bob"
16 *-* PARSE VAR tst_name new_name .
    >>> "bob"
    >.> ""
17 *-* IF new_name = old_name
    >>> "0"
20 *-* ELSE
    *-* DO
21 *-* NOP
22 *-* END
***
```



TRACE S

```
11 *-* TRACE S
12 *-* "test"
13 *-* old_name = "Fred"
14 *-* SAY "Please enter name : "
15 *-* PARSE PULL tst_name
16 *-* PARSE VAR tst_name new_name .
17 *-* IF new_name = old_name
   *-* THEN
   *-* DO
18 *-*   SAY "Hello Fred"
20 +++   ELSE
Error running TRACE, line 20: Unexpected THEN or ELSE
***
```



5-17

The Software That Manages eBusiness™

? And ! In the TRACE Instruction

- ? - Toggles interactive debugging
 - display change variables
 - execute instructions
 - used with R and I e.g TRACE ?I
- ! - Toggles host command execution inhibit
 - will not execute TSO commands
 - use with C, R and I, or by itself.



5 - 18

The Software That Manages eBusiness™

Error Trapping

- ? - Terminate or continue
- SIGNAL instruction
 - SIGNAL ON NOVALUE
 - SIGNAL ON SYNTAX
 - SIGNAL ON FAILURE
 - SIGNAL ON ERROR
 - SIGNAL ON HALT
- Ends with an EXIT or RETURN - may not pass back any information



5-19

The Software That Manages eBusiness™

ERROR - a TSO command ends with a non-zero return code

FAILURE - a TSO command ends with a negative return code

HALT - the terminal attention key is pressed and a HI (Halt Interpretation) command is entered

NOVALUE - an uninitialized variable is used in an expression or in a PARSE template

SYNTAX - a REXX interpretation syntax error happens

Error Trapping

- Special variables assigned in TSO/E REXX.
 - SIGL
 - variable holds line number in error
 - SOURCELINE
 - used to extract the relative line number of a line within the source for current REXX exec.
 - ERRORTXT(RC)
 - used to extract from REXX the error message that is associated with a particular REXX error number.
 - CONDITION
 - used to retrieve the setting information for the currently trapped REXX condition.



Error Trapping

```
SIGNAL ON NOVALUE
IF new_name = old_name THEN DO
  SAY "Hello Fred"
END
EXIT

NOVALUE:
  SAY "Variable" CONDITION("D") has not been initialised on
  SAY "Line : "SIGL
  SAY "Source code : "SOURCELINE(SIGL)
```

```
Variable NEW_NAME HAS NOT BEEN INITIALISED ON
Line : 12
Source code : IF new_name = old_name THEN DO
***
```



Work Section 5.1

- Re-Write Work Section 1.2
- Test the different TRACE options.
 - A
 - C
 - E
 - F
 - I
 - L
 - N
 - O
 - R
 - S



5 - 22

The Software That Manages eBusiness™

Work Section 5.2

- Execute the following Program using Trace ?i To see how REXX treats tails in compound variables.

```
TRACE ?I
day = "Tuesday"
month.day = "May"
tuesday = "Tuesday"
SAY month.tuesday
```



5 - 23

The Software That Manages eBusiness™

Additional program

- Try using the TRACE() function instead
- e.g.
- TRACE(I)

- What is the difference ?

- Write a program using SIGNAL ON NOVALUE.
 - Use a SAY statement to display a variable which has not been initialised and use the NOVALUE label to give an appropriate message



5 - 24

The Software That Manages eBusiness™

Looping Instructions

Section 6



The Software That Manages eBusiness™

DO

```
loop_counter = 0
DO 5
  loop_counter = loop_counter + 1
  SAY loop_counter
END
```

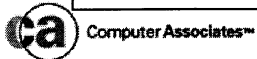
```
1
2
3
4
5
***
```



DO FOREVER

```
DO FOREVER
  SAY "Your name please : "
  PARSE UPPER EXTERNAL name
  PARSE VAR name fore_name .
  IF fore_name = "BOB" THEN DO
    EXIT
  END
END
```

```
Your name please :
sdfgfd
Your name please :
fg
Your name please :
BOB
***
```



Test Exercise 61

- Write a REXX program to loop 5 times and show even numbers to the screen only.

```
Count is : 2  
Count is : 4  
***
```



DO count = 1 TO n BY n

```
DO loop_counter = 1 TO 5 BY 1
  SAY loop_counter
END
```

```
1
2
3
4
5
***
```



Test Exercise 62

- Write a REXX program to count to 10 showing only every third number.

```
1  
4  
7  
10  
***
```



DO WHILE

```
loop_counter = 0
DO WHILE loop_counter < 5
  loop_counter = loop_counter + 1
  SAY loop_counter
END
```

```
1
2
3
4
5
***
```



DO UNTIL

```
loop_counter = 0
DO UNTIL loop_counter < 5
  loop_counter = loop_counter + 1
  SAY loop_counter
END
```

```
1
***
```



ITERATE

```
DO loop_counter = 1 TO 5
  IF loop_counter = 3 THEN DO
    ITERATE
  END
  SAY loop_counter
END
```

```
1
2
4
5
***
```



LEAVE

```
DO loop_counter = 1 TO 5
  IF loop_counter = 3 THEN DO
    LEAVE
  END
  SAY loop_counter
END
```

```
1
2
***
```



6 - 10

The Software That Manages eBusiness™

Looping with Compound variables

```
DO loop_counter = 1 TO 3
  PARSE UPPER EXTERNAL new_name
  full_name.loop_counter = new_name
END
DO test_counter = 3 TO 1 BY -1
  SAY full_name.test_counter
END
```

```
BOB
GARY
FRED
FRED
GARY
BOB
***
```



6-11

The Software That Manages eBusiness™

Work Section 6.1

- Write a REXX program which will:
 - ask for 10 numbers from the user
 - assign the numbers to compound variables
 - output each variable and its value
 - output the average of the 10 variables

```
Please enter a number:
```

```
1
```

```
Please enter a number:
```

```
2
```

```
Please enter a number:
```

```
3
```

```
Please enter a number:
```

```
4
```

```
Please enter a number:
```

```
5
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
Average = 3
```

```
***
```



Work Section 6.2

- Re-write Work section 2.1 to allow the user to enter their name as many times as they want.
- Stop the program if they don't enter any more names.



6 - 13

The Software That Manages eBusiness™

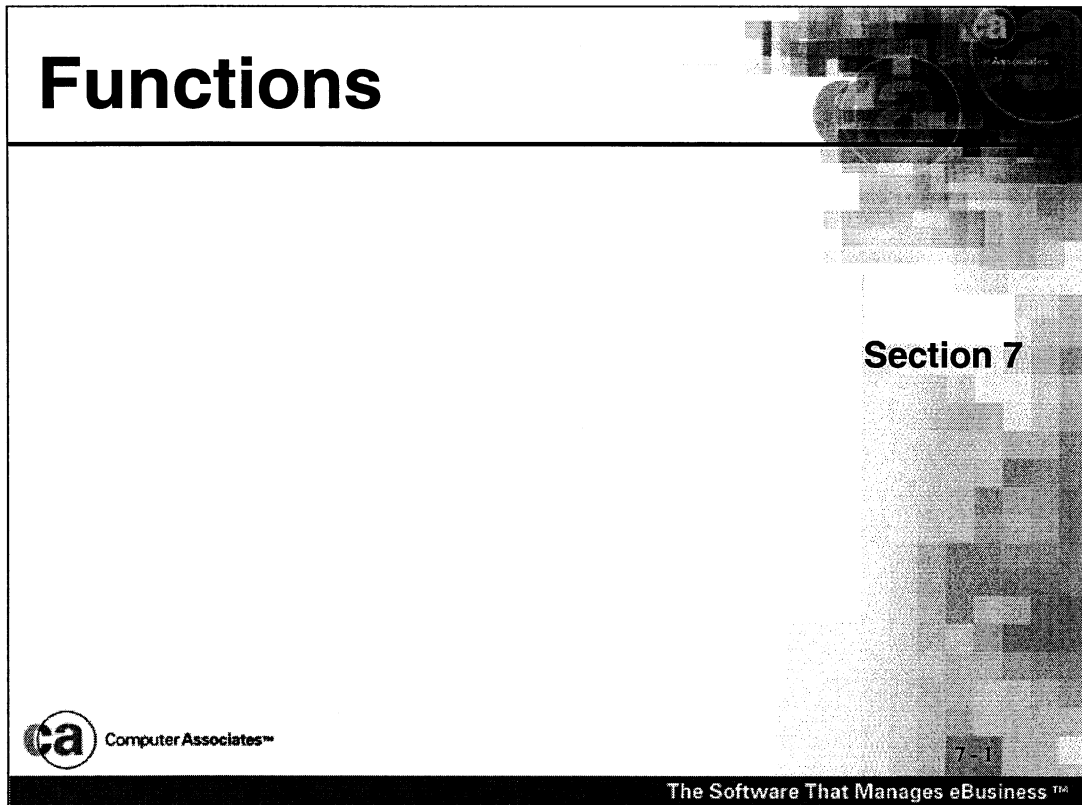
Additional Program

- Re-write program 6.1 and find the highest number entered.
- Also display the numbers in reverse order.



6 - 14

The Software That Manages eBusiness™




What is a function?

- A pre-written sub-routine.
- A Function returns a value.
- The function name is suffixed with brackets, which are used for any arguments.
- REXX has a number of supplied functions.



DATATYPE

<pre>SAY DATATYPE("AA", A) SAY DATATYPE("1", B) SAY DATATYPE("A", L) SAY DATATYPE("Aa", M) SAY DATATYPE("1", N) SAY DATATYPE("a", U) SAY DATATYPE("1.2", W) SAY DATATYPE("1", X) SAY DATATYPE("1", B) SAY DATATYPE("1", B) SAY DATATYPE("1", B) SAY DATATYPE("1", B) SAY DATATYPE("1", B) SAY DATATYPE("1", B) SAY SAY DATATYPE("1") SAY DATATYPE("A")</pre>	<pre>1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 NUM CHAR ***</pre>
--	---


Computer Associates™

The Software That Manages eBusiness™

A for Alphanumeric, returns a 1 if 'string' contains upper or lower case letters or the numbers 0 through 9.

N for Numeric, returns a 1 if 'string' is a valid number acceptable to REXX

W for Whole number, returns a 1 if 'string' is a whole number acceptable to REXX under the current setting of NUMERIC DIGITS.

L for Lowercase, returns a 1 if 'string' contains ONLY lowercase letters in the range A-Z.

U for Uppercase, returns a 1 if 'string' contains ONLY uppercase letters in the range A-Z.

M for Mixed case, returns a 1 if 'string' contains either upper or lowercase letters in the range A-Z.

S for Symbol, returns a 1 if 'string' contains only the characters that are valid for forming REXX symbols.


B for Bitstring, returns a 1 if 'string' contains ONLY 0's and 1's.

X for Hexadecimal, returns a 1 if 'string' contains valid hexadecimal characters in the ranges a-f, A-F, 0-9, or blank. A 1 is also returned if 'string' is a null (zero length) string.

POS

```
SAY POS(".", "CLCS.IULC00.REXX")
line = "/******REXX*****/"
SAY POS("REXX", line)
```

```
5
15
***
```

 Computer Associates™

7-4

The Software That Manages eBusiness™


POS(substring,string{,startloc})

Returns the position of one string, in another.

LEFT

```
SAY LEFT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY LEFT(line, 7)
```

```
RE
IST510I
***
```

 Computer Associates™

7-6

The Software That Manages eBusiness™


`LEFT(string, length{, pad})`

Returns a string of length, containing the leftmost length.

RIGHT

```
SAY RIGHT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY RIGHT(line, 7)
```

```
XX
NG ONLY
***
```

 Computer Associates™

The Software That Manages eBusiness™

`RIGHT(string, length{, pad})`

Returns a string of length, containing the rightmost length.

STRIP

```
SAY STRIP("  REXX  ")
line = "0.12000000"
SAY STRIP(line, "T", 0)
```

```
REXX
0.12
***
```



The Software That Manages eBusiness™

`STRIP(string[, {option}[, char]])`

Returns string with leading or trailing characters or both removed.

SUBSTR

```
SAY SUBSTR("REXX PROGRAMMING", 4, 3)
line = "IST510I TESTING ONLY"
SAY SUBSTR(line, 7, 1)
```

```
X P
I
***
```



The Software That Manages eBusiness™

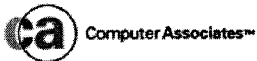
`SUBSTR(string, n{, {length}{, pad}})`

Returns the substring of string that begins at the nth character and is of length

ABBREV

```
SAY ABBREV("REXX PROGRAMMING", "REXX P")
line = "CON"
IF ABBREV("CONFIRM", line, 1) = 1 THEN DO
    SAY "OK"
END
```

```
1
OK
***
```



7 - 10

The Software That Manages eBusiness™


`ABBREV(string, prefix{, length})`

Returns 1 if info is equal to the leading characters of information.

TRANSLATE

```
SAY translate("23.12.2000", "/", ".")
line = "CLCS.IULC00.REXX"
SAY TRANSLATE(line, " ", ".")
```

```
23/12/2000
CLCS IULC00 REXX
***
```



Computer Associates™

7-11

The Software That Manages eBusiness™

`TRANSLATE(string{, {outtab}{, {intab}{, pad}}})`

Returns string with each character translated to another character.

DELSTR

```
SAY DELSTR("CLCS.IULC00.REXX", 6, 6)
line = "/******REXX*****/"
SAY DELSTR(line, 15, 4)
```

```
CLCS..REXX
/******/
***
```



7-12

The Software That Manages eBusiness™


`DELSTR(string, n, length)`

Returns string after deleting the substring that begins at the nth character and is of length characters.

INSERT

```
SAY INSERT("IULC00", "CLCS..REXX", 5)
line = "/******REXX***/"
SAY INSERT("REXX", line, 15)
```

```
CLCS.IULC00.REXX
/******REXX***/
***
```


Computer Associates™

7-13
The Software That Manages eBusiness™


`INSERT (ins, target {, {n} {, {length} {, pad} } })`

Inserts the string new, padded or truncated to length length, into the string target after the nth character.

OVERLAY

```
SAY OVERLAY("IULC22", "CLCS.IULC00.REXX", 6)
line = "/*****TEST*****/"
SAY OVERLAY("REXX", line, 15)
```

```
CLCS.IULC22.REXX
/*****REXX*****/
***
```


Computer Associates™

7 - 14
The Software That Manages eBusiness™

OVERLAY(new, target(, {n}{, {length}{, pad}}))

Returns the string target, which, starting at the nth character, is overlaid with the string new, padded or truncated to length length.

The slide is titled "CENTRE" and features a background image of a computer monitor. It contains two code blocks. The first block shows REXX code: `SAY CENTRE("CLCS.IULC00.REXX", 50)`, `line = "REXX"`, and `SAY CENTRE(line, 79, "**")`. The second block shows the output: `CLCS.IULC00.REXX` followed by `*****REXX*****` on the next line, and `***` on the third line. The slide also includes the Computer Associates logo and the slogan "The Software That Manages eBusiness™".

`CENTRE(string, length{ , pad})`

Returns a string of length length with string centered in it, with pad characters added as necessary to make up length.


SPACE

```

SAY SPACE("This      is      the      REXX Course", 1)
line = "CLCS      IULCC00  REXX"
SAY SPACE(line, 1, ".")
SAY SPACE("A B C D E F G", 0)
SAY SPACE("A B C", 5)
    
```

```

This is the REXX Course
CLCS.IULCC00.REXX
ABCDEFGG
A      B      C
***
    
```

 Computer Associates™

7 - 16

The Software That Manages eBusiness™

SPACE(string{, {n}{, pad}})

Returns the blank-delimited words in string with n pad characters between each word.

FORMAT

```
SAY FORMAT("12000", 10)
line = "3.5"
SAY FORMAT(line, 10)
SAY FORMAT("124.5656", 10, 2)
SAY FORMAT("17591.73",,,2,2)
```

```
12000
      3.5
     124.57
1.759173E+04
***
```



`FORMAT (number { , {before} } { , {after} })`

Returns number, rounded and formatted.

COMPARE

```
SAY COMPARE("Neville", "Neville")
line = "/*****REXX*****/"
SAY COMPARE(line, "TEST_STRING")
```

```
0
1
***
```



7 - 18

The Software That Manages eBusiness™

`COMPARE(string1, string2(, pad))`

Returns 0 if the strings, string1 and string2, are identical. Otherwise, returns the position of the first character that does not match.

COPIES

```
SAY COPIES("REXX ", 5)
line = "GREAT"
SAY COPIES(line, 3)
```

```
REXX REXX REXX REXX REXX
GREATGREATGREAT
***
```



7-19

The Software That Manages eBusiness™

`COPIES(string, n)`

Returns n concatenated copies of string.

LENGTH

```
SAY LENGTH("CLCS.IULC00.REXX")  
line = "/******REXX*****/"  
SAY LENGTH(line)
```

```
16  
35  
***
```



7 - 20

The Software That Manages eBusiness™

LENGTH(string)

Returns the length of string.

REVERSE

```
SAY REVERSE("CLCS.IULC00.REXX")  
line = "Hello the REXX course"  
SAY REVERSE(line)
```

```
XXER.00CLUI.SCLC  
esruoc XXER eht olleH  
***
```



The Software That Manages eBusiness™

`REVERSE(string)`

Returns string, swapped end for end.

DELWORD

```
SAY DELWORD("Welcome to REXX", 2)
line = "Welcome to REXX"
SAY DELWORD(line, 2, 1)
```

```
Welcome
Welcome REXX
***
```

Computer Associates™

7 - 22

The Software That Manages eBusiness™

`DELWORD(string, n{, length})`

Returns string after deleting the substring that starts at the nth word and is of length blank-delimited words.

SUBWORD

```
SAY SUBWORD("Welcome to REXX programmers", 2)
line = "Welcome to REXX programmers"
SAY SUBWORD(line, 2, 2)
```

```
to REXX programmers
to REXX
***
```



7-23

The Software That Manages eBusiness™

`SUBWORD(string, n{, length})`

Returns the substring of string that starts at the nth word, and is up to length blank-delimited words.

The slide features a title 'WORD' in the top left. Below it, a code block contains the following REXX code:

```
SAY WORD("Welcome to REXX programmers", 2)
line = "Welcome to REXX programmers"
SAY WORD(line, 3)
```

A second code block below shows the output:

```
to
REXX
***
```

The slide includes the Computer Associates logo in the bottom left and the slogan 'The Software That Manages eBusiness™' in the bottom right. A page number '7 - 24' is visible in the bottom right corner of the slide content area.

`WORD(string,n)`

Returns the nth blank-delimited word in string or returns the null string if fewer than n words are in string.

The screenshot shows a presentation slide with a dark background and a light-colored text box. The title 'WORDINDEX' is at the top left. The code block contains three lines of REXX code. The output block shows the results of the code execution. The slide also features the Computer Associates logo and tagline at the bottom.

```
WORDINDEX
```

```
SAY WORDINDEX("Welcome to REXX programmers", 3)
line = "Welcome to REXX programmers"
SAY WORDINDEX(line, 4)
```

```
12
17
***
```

Computer Associates™

7-25

The Software That Manages eBusiness™

`WORDINDEX(string, n)`

Returns the position of the first character in the nth blank-delimited word in string or returns 0 if fewer than n words are in string.

WORDLENGTH

```
SAY WORDLENGTH("Welcome to REXX programmers", 2)
line = "Welcome to REXX programmers"
SAY WORDLENGTH(line, 3)
```

```
2
4
***
```



7 - 26

The Software That Manages eBusiness™

`WORDLENGTH(string, n)`

Returns the length of the nth blank-delimited word in string or returns 0 if fewer than n words are in string.

WORDS

```
SAY WORDS("Welcome to REXX programmers")  
line = "Welcome REXX programmers"  
SAY WORDS(line)
```

```
4  
3  
***
```



The Software That Manages eBusiness™


WORDS(string)

Returns the number of blank-delimited words in string.

Arithmetic Functions

```
SAY ABS(-32)
SAY ABS(32)
SAY MIN(234, 3245, 3, 234)
SAY MAX(234, 3245, 3, 234)
SAY RANDOM(1, 49)
SAY SIGN(-32)
SAY TRUNC(213.1487876, 2)
```

```
32
32
3
3245
9
-1
213.14
***
```

 Computer Associates™

7-28

The Software That Manages eBusiness™

ABS (number)

MIN (number { , number } ...)

MAX (number { , number } ...)


RANDOM ({min} { , {max} { , seed } }

SIGN (number)

TRUNC (number { , n })

DATE

<pre> SAY DATE() SAY DATE("B") SAY DATE("C") SAY DATE("D") SAY DATE("E") SAY DATE("J") SAY DATE("M") SAY DATE("O") SAY DATE("S") SAY DATE("U") SAY DATE("W") </pre>	<pre> 3 Feb 2000 730152 34 34 03/02/00 00034 February 00/02/03 2000203 02/03/00 Thursday *** </pre>
---	---


Computer Associates™

7 - 29
The Software That Manages eBusiness™

C as in Century, returns number of days into the current century in the form 'dddd' with leading zeros stripped.

D for Days, returns number of days into the current year in the form 'ddd' with leading zeros stripped.

U returns date in USA format, 'mm/dd/yy'

E for European dates, returns the date in the form 'dd/mm/yy'

J returns a Julian date in the form 'yyddd'

M returns the full name of the current month (e.g., December)


O as in 'Ordered' date, returns the date in the form 'yy/mm/dd', allowing sorting in chronological date sequence

S returns an alternative sort-format date in the form 'yyyymmdd'

W returns the day of the week (e.g. 'Tuesday', 'Sunday', etc.)

TIME

<pre> SAY TIME() SAY TIME("E") SAY TIME("H") SAY TIME("L") SAY TIME("M") SAY TIME("R") SAY TIME("S") </pre>	<pre> 05:18:59 0 5 05:18:59.066418 318 0.000432 19139 *** </pre>
---	--


Computer Associates™

7 - 30

The Software That Manages eBusiness™

E - Causes TIME to return the elapsed time since a previous TIME('R') in seconds and microseconds.

R - Causes TIME to return the elapsed time since a previous TIME('R') in seconds and microseconds. And causes the elapsed time to be Reset.

H - Causes TIME to return the number of hours since midnight on the same day in the form 'hh', with no leading zeros

L - Causes TIME to return the current time of day in the form hh:mm:ss.uuuuuu, where 'uuuuuu' is a fractional number of seconds, in microseconds (millionths of seconds)

M - Causes TIME to return the current time of day as the number of minutes since midnight on the same day in the form mmmm, with no leading zeros

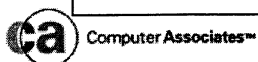
S - Causes TIME to return the current time of day as the number of seconds since midnight on the same day in the form sssss, with no leading zeros

Work Section 7.1

- Write a REXX program which will:
 - Format a title in the the centre of the screen and underlined.
 - Show today's date in the format : mm/dd/yy
 - Show the time in the format : hh:mm:ss
 - Show the date in the format DD-MM-YY

```
Function Program
=====

The american formatted date is : 02/03/00
This program was executed at : 05:42:57
The european date : 03-02-00
***
```



Work Section 7.2

- Write REXX program to prompt for a Name and check that is your name, the program can accept any way of writing your name. If it is not your name loop round until your name is entered or the word "STOP"
 - E.g. FRED SMITH or FRED or FRED S
- Ask for a selection of 4 numbers.
 - Show the highest number
 - Show the lowest Number



The Software That Manages eBusiness™

```
Please enter your name :
mick smith
Please enter your name :
mike de
Please enter four numbers
1
2
3
6
The highest number is : 6
The lowest number is : 1
***
```


Additional Program

- Write a REXX program to play a guess the number game.

```
                                Number game
                                =====

Please Guess the number (1-100) :
50
Too high
Please Guess the number (1-100) :
25
Too high
Please Guess the number (1-100) :
10
Too high
Please Guess the number (1-100) :
5
Too low
Please Guess the number (1-100) :
7
Hurrah you guessed the number in 5 guesses.
***
```



7-33

The Software That Manages eBusiness™

Section end

ca Computer Associates™

7 - 34

The Software That Manages eBusiness™