4 The REXX functions

Functions play an important role in REXX. In particular, the functions for word and string processing. When you have worked through the next chapter, you might be amazed about what you can do with the REXX functions. So, have fun reading about the REXX functions! The REXX language we can use on z/OS TSO/E consists of two blocks of language elements:

– The REXX commands, as we learned in chapter 3 The REXX commands on page 25.

– The large stock of standard functions that belong to the REXX language and which we will discuss in this chapter.

We now turn to the second part, the REXX functions. I will not discuss any function that is described in the brochure TSO/E REXX reference, but only those that are in my opinion often needed in practice. How functions are called, I have explained already in the section 3.3 CALL – Call other programs on page 35. There we already talked about the topics parameter input and return of the results.

The REXX functions can be divided into groups by application area. I have made the following division into groups for the presentation of the REXX functions in this book.

Table 4.1: REXX function groups

Function group	On page
General Functions	64.
Arithmetic Functions	72.
Comparison Functions	74.
Conversion Functions	77.
Formatting Functions	82.
String Functions	85.
Word Functions	90.
System Functions	92.

The number of available standard functions is larger than the number of instructions in the REXX language. This lets you know how important the knowledge of the functions in REXX is. Through internal and external programs, you can easily create your own, new features and thereby extend your pool of available functions arbitrarily.

General rules for the use of functions:

– The REXX interpreter recognizes a function call when parentheses follow the function name.

– Some functions return a value even if no parameters have been entered. In this case, the parentheses must still be set. Then they remain empty.

– The left parenthesis must always follow immediately the function name without any blanks, otherwise the function name will be assumed as REXX command. If this rule is not adhered, then a program abort occurs.

In the headings of the following functional descriptions, I will omit the parentheses respectively.

4.1 GENERAL FUNCTIONS

4.1.1 ADDRESS – Get the active host command environment

This function returns the name of the current host command environment. No parameters can be entered.

Function:

Returns the currently active host command environment.

Format:

name = address()

A host command environment, whose name was written by the ADDRESS function to a variable, can only be restored by using the following commands.

– By setting the variable in surrounding parentheses.

– By using the REXX VALUE function.

The following example shows the two variants:

```
envir = address()  /* Save the current active HCE in envir  */
/* processing */.....
address (envir)    /* These both commands restores the HCE */
address value envir /* saved in envir                       */
```

HCE → Host Command Environment

See also the section 3.1 ADDRESS – Connection to the subsystems on page 25.

4.1.2 ARG – Input parameter test or take

The ARG function has three application forms:

– Determining the number of input parameters.

– Testing for the presence of certain input parameters.

– Taking single or all the parameters.

Format:

ARG (n,option)

n	Means the nth parameter. If specifying this option, the contents of the nth parameter will returned. If this parameter is not set, a null string will be returned.
option	**e** Returns 1 if the nth parameter is available. Otherwise, 0 is returned. **o** Returns 1 if the nth parameter is **not** available. Otherwise, 0 is returned.

```
/* Call name (no arguments) */
ARG()          ->    0
ARG(1)         ->    ''
ARG(2)         ->    ''
ARG(1,'e')     ->    0
ARG(1,'O')     ->    1
/* Call name 'a',,'b'; /* (three arguments)  */
na = ARG()     ->    3
a1 = ARG(1)    ->    'a'
a2 = ARG(2)    ->    ''
a3 = ARG(3)    ->    'b'
ARG(n)         ->    '' /* for n >= 4 */
ARG(1,'e')     ->    1
ARG(2,'E')     ->    0
ARG(2,'O')     ->    1
ARG(3,'o')     ->    0
ARG(4,'o')     ->    1
```

4.1.3 DATE – Date functions

The DATE function returns the current date in many different forms. Moreover, you can perform with the DATE function also date calculations.

– Return of the current date in different forms.

– Calculation of date values.

On the following pages, I will explain these two variants in detail.

1. Return the current date

Returns today's date back in different forms.

Format:

today_date = DATE(format)

It returns the current date in the notation defined in the format. The possible entries for format and the resulting return values are summarized in the following table. Only the first letter is entered to define the format.

Table 4.2: Parameters of the DATE function

Format	Returned value
Base	Days since January 1, 0001 without the present day. The format **wd = date("B") // 7**
	can be used to determine the current day of the week. 0 is Monday and 6 is Sunday.
Century	Days since January 1 of the current century, including today.
Days	Days since January 1 of the current year, including today.
European	Date in European format: DD/MM/YY
Julian	Date in the form: YYDDD this means Julian Day.
Month	English name of the month in upper and lower case. For example, January.
Normal	Date in the format: DD Mon YYYY. Mon is the short form of month: Jan, Feb, Mar, etc. This is the **default** setting, if no parameter is entered.
Ordered	Date in the format: YY/MM/DD.
Standard	Date in the format: YYYYMMDD.
Usa	Date in the American format: MM/DD/YY.
Weekday	English weekday. Monday, Tuesday, etc.

Example DATE output:

```
DATE()     --> 29 Apr 2004
DATE('B') --> 731699
DATE('C') --> 1581
DATE('D') --> 120
DATE('E') --> 29/04/04
DATE('J') --> 04120
DATE('M') --> April
DATE('N') --> 29 Apr 2004
DATE('O') --> 04/04/29
DATE('S') --> 20040429
DATE('U') --> 04/29/04
DATE('W') --> Thursday
```

2. Date calculations using the DATE function

With the DATE function, you can perform some date conversion to another format and date calculations.

Format:

result = DATE(format1,input,format2) Rule:

Rule:

To demonstrate these capabilities of the DATE function, I wrote and performed a small REXX procedure. See the result here:

```
Todays date = 29 Apr 2004
Calculation: Today minus 200 days, result in N (DD Mon YYYY)
Statement  : date("N",date("B") - 200,"B")
Result     : 12 Oct 2003

Calculation: S date plus 100 days, result in S (YYYYMMDD)
Statement  : date("S",date("B",20040101,"S") + 100,"B")
Result     : 20040410

Calculation: Today plus 360 days,  result in E (DD/MM/YY)
Statement  : date("E",date("B") + 360,"B")
Result     : 24/04/05

Conversion : S format (YYYYMMDD) in N format (DD Mon YYYY)
Statement  : DATE(,"20020609","S")
Result     : 9 Jun 2002

Conversion : N format (DD Mon YYYY) in B format
Statement  : DATE("B","25 Sep 2001")
Result     : 730752

Conversion : C format (DDDD) in N format (DD Mon YYYY)
Statement  : DATE("N","1000","C")
Result     : 26 Sep 2002

Conversion : O format (YY/MM/DD) in U format (MM/DD/YY)
Statement  : DATE("U","03/08/15","O")
Result     : 08/15/03

Conversion : J format (YYDDT) in E format (DD/MM/YY)
Statement  : DATE("E","01101","J")
Result     : 11/04/01
```

Note:

For conversions and calculations, formats C and J cannot be used as a result. This is a shame, because conversions into the J format are often necessary in practice.

4.1.4 TIME – Time functions

The TIME function returns the current date in many different forms. Moreover, you can perform with the TIME function also date calculations.

Function:

The TIME function provides two ways to use it:

- Returns the current time of day.
- Starting and stopping the difference timing measurement.

Format:

Time = TIME (format)

The following table shows which options can be set with format and which results TIME then provides.

Table 4.3: Options of the TIME function

Format	Return value
Civil	Time of day in the form HH:MMxx, where xx can be either am or pm. This is for American time information.
Elapsed	Result is the time in the form SSSSSSSSS.UUUUUU, where seconds are before the decimal point and the decimals of seconds behind. With this form, the stopwatch is started. Each subsequent call with this form gives the elapsed time back since the first call.
Hours	Returns the hour of day in one or two digits.
Long	Returns the time in the format HH:MM:SS.UUUUUU. The digits after the decimal point are microseconds.
Minutes	Returns the minutes that have elapsed since midnight.
Normal	Returns the time in the form HH:MM:SS. This is the default.
Reset	Specifies the time back since the start or the last reset of the timer and resets the stopwatch to

Seconds	Returns the seconds that have elapsed since midnight.

Examples of the return time of day

```
TIME()     --> 17:40:23
TIME('C') --> 5:40pm
TIME('H') --> 17
TIME('L') --> 17:40:23.656081
TIME('M') --> 1060
TIME('N') --> 17:40:23
TIME('S') --> 63623
```

Examples of the time measurement

```
TIME('E') --> 0
TIME('R') --> 0.000530
```

Between the call with the format **E** and the call with the format **R** 530 microseconds have elapsed.

4.1.5 QEUED – Number of records in the data stack Function:

Function:

Returns the number of records in the TSO data stack.

Format:

Number = QUEUED()

Returns the number of records currently contained in the data stack. No parameters can be entered. A return value of zero indicates an empty data stack.

4.1.6 SOURCELINE – Return a program line

Function:

– Returns the number of lines of source code.

– Returns the text of a line from the source code of the program.

Format:

Number = SOURCELINE()

Returns the number of lines of the source code.

Text = SOURCELINE(n)

Returns the contents of the nth row. n must be entered.

Example:

With the following DO loop, you can run through all lines of program code:

```
do i = 1 to sourceline()
line = sourceline(i)
/* Processing the source line */
end i
```

4.1.7 USERID – Return the TSO user ID

Function:

Return of the TSO user ID.

Format:

user = userid()

4.1.8 VALUE – Create variable names dynamically Function:

Function:

Generating a value, which consists of a combination of variables and constants.

Format:

value = VALUE("XYZ"var)

The VALUE function dynamically generates a value consisting of constant and variable parts. The variables are replaced by their contents. This command works similarly to the INTERPRET instruction. However, while INTERPRET always constitutes the entire command and then executes it, returns VALUE only the assembled value.

Example:

The edit macro #TSOB from the SMART ISPF utilities submits REXX

can occur. The SMART ISPF utilities program SPROFVAR stores this information into the ISPF user profiles. The organization of the variable is as follows:

The profile variables SYSEXEC and STEPLIB each contains the number of existing DSNs for this type that are to be included in the batch job. The corresponding DSNs are then in the variables SYSEXECn and STEPLIBn, where n is from 1 to content of SYSEXEC and STEPLIB respectively.

The following commands define the variables. That is their content in the profile pool.

```
sysexec  = 4
steplib  = 3
sysexec1 = "PROX.LANZ.REXX"
sysexec2 = "PROX.LOGON.REXX"
sysexec3 = "SMQP.USER.CEXEC"
sysexec4 = "SMQP.PROD.CEXEC"
steplib1 = "PROX.LOGON.LOAD"
steplib2 = "DB2P.ALIAS.DSNLOAD"
steplib3 = "SMQP.LOAD"
```

Subsequently, the program section from the edit macro #TSOB that reads the variables from the profile and includes them in the JCL:

```
/*********************************************************************/
/* Reading the environment variables                                 */
/*********************************************************************/
"ISPEXEC CONTROL ERRORS RETURN"
"ISPEXEC VGET (SYSEXEC STEPLIB",
              "ACCOUNT JOBCLASS MSGCLASS) PROFILE"
if rc > 0 then do
   zedlmsg = " Error: Variable could not correctly read from ",
             "profile pool. Procedure exits."
   if zerrlm <> "ZERRLM" then zedlmsg = zedlmsg strip(zerrlm)
   address "ISPEXEC" "SETMSG MSG(ISRZ001) COND"
   exit
end
/*********************************************************************/
/* Assemble the batch job                                            */
/*********************************************************************/
jobname  = userid()jobsuff()
queue "//"jobname" JOB ("account"),'@TSOB: "mem"',CLASS="jobclass","
queue "//   NOTIFY="userid()",MSGLEVEL=(0,0),MSGCLASS="msgclass","
queue "//   COND=(0,LT),TIME=100,USER=SQMTP"
queue "//"copies(**",68)
queue "//*  TSO-BATCH FROM: "dsn"("mem")"
queue "//"copies(**",68)
queue "//TSOBATCH EXEC PGM=IKJEFT01,REGION=80M,DYNAMNBR=99"
if steplib > 0 then do
   "ISPEXEC VGET (STEPLIB1) PROFILE"
   if rc > 0 then call ispf_error rc "VGET STEPLIB1"
   queue "//STEPLIB  DD  DISP=SHR,DSN="steplib1
   do i = 2 to steplib
      "ISPEXEC VGET (STEPLIB"i") PROFILE"
      if rc > 0 then call ispf_error rc "VGET STEPLIB"i
      queue "//         DD  DISP=SHR,DSN="value("steplib"i)
   end i
end
queue "//SYSEXEC  DD  DISP=SHR,DSN="dsn
if sysexec > 0 then do i = 1 to sysexec
   "ISPEXEC VGET (SYSEXEC"i") PROFILE"
   if rc > 0 then call ispf_error rc "VGET SYSEXEC"i
   queue "//         DD  DISP=SHR,DSN="value("SYSEXEC"i)
end i
```

The rest of the procedure is suppressed.

4.2 ARITHMETIC FUNCTIONS

In the following sections, I will discuss functions relating to various ways of treatment and control of arithmetic operations.

4.2.1 ABS – absolute value of a number

Function:

Returns the absolute value of a number.

Format:

result = ABS(number)

This function returns the unsigned absolute value of a number. The format of the returned number is obtained from the currently valid NUMERIC options.

Examples:

```
ABS(-2/3)   0.666666667
ABS('-0.307')   0.307
```

4.2.2 DIGITS, FORM, FUZZ – Query options for arithmetic operations

These three functions query the values that apply just for performing arithmetic expressions. This information can also be obtained by the PARSE NUMERIC var command.

Example:

```
numeric digits 12
numeric fuzz   3
parse numeric nums
dig = digits()
fuz = fuzz()
```

Here the results when calling this procedure:

```
Values from PARSE       = 12 3 SCIENTIFIC
Values from functions   = 12 3 SCIENTIFIC
```

4.2.3 MIN, MAX – Minimum and maximum value

These functions return the minimum or maximum value of a set of numbers that are entered as a single parameter. In a parameter list, a maximum of 20 values can be entered. However, since these functions could be interleaved, a maximum of 20 x 20 = 400 numbers are possible in one function call.

4.2.4 RANDOM – Generate random numbers

Function:

This function generates a random number within a lower and an upper limit.

Format:

number = random(start,end[,ind])

start	lower limit of the number space
end	upper limit of the number space
ind	base number by which one can achieve the reproducibility.

Example:

In this example, it is shown that with repeated calls always the same random numbers are received when using the same number for option ind.

Program 4.1: RAMDOM – Generate random numbers

```
/* DOC: REXX RANDOM                                          */
/* DOC: Example for the generation of random numbers*/
/* © FRANZ LANZ 2015                                         */
/************************************************************/
nnn = ""
do 10;          nnn = nnn random(1,1000); end
say nnn
txt = random(1,1000,12)
do i = 1 to 10; txt = txt random(1,1000); end i
say txt
seq = random(1,12,3)
do i = 1 to 10; seq = seq random(1,12);   end i
say seq
txt = random(1,1000,12)
do i = 1 to 10; txt = txt random(1,1000); end i
say txt
seq = random(1,12,3)
do i = 1 to 10; seq = seq random(1,12);   end i
say seq
```

This is the printout of the above program:

```
311 886 265 230 967 115 390 676 577 161
976 509 127 16 332 637 364 595 134 852 333
3 8 2 6 5 11 2 4 12 5 9
976 509 127 16 332 637 364 595 134 852 333
3 8 2 6 5 11 2 4 12 5 9
```

As you can see, in rows 2 and 4 plus 3 and 5 respectively appear the same numbers, because in the corresponding RANDOM calls a base number was specified. It is irrelevant what base number you choose.

4.2.5 SIGN – Return of the sign

Function:

Returns the sign of a number.

Format:

value = sign(number)

SIGN returns the sign of the number specified by the following rules:

```
number > 0 →  value=  1
number = 0 →  value=  0
number < 0 →  value= -1
```

4.3 COMPARISON FUNCTIONS

Besides the possibility of comparisons using the IF command, there are some functions that you can use to perform comparisons in REXX. The most important function is DATATYPE, because without this function plausibility checks of numerical data could not be performed.

4.3.1 COMPARE – Compare texts

Function:

This function returns a zero if the two texts are equal. If the texts are not equal

result = COMPARE(text1,text2[,filler])

text1,text2	The texts to compare.
filler	The shorter of the two texts is expanded with the filler before comparison. The default character is a blank.

Note:

I have never needed this feature because this function can be carried out with the normal logical text comparisons operands, too. However, if you want to find the position from which the inequality begins, COMPARE is very useful.

4.3.2 DATATYPE – Determine data type

Function:

This is a very important function. It is always required to make sure before an arithmetic operation that all participating operands are of type NUM.

Format:

result = datatype(text,[type])

result	It is **NUM** if text contains a number that can be used in arithmetic operations.
	It is **CHAR** in all other cases
text	This is the text, which shall be examined. When only this parameter is specified, then the function returns the values NUM or CHAR.
type	When type is specified, then the function returns 1 if the text contains only elements of the entered **type** property, otherwise 0.

With type, the following properties can be queried:

Table 4.4: DATATYPE type options

Type	Properties
A	Text consists entirely of alphanumeric characters (a-z, A-Z, 0-9).
B	Text consists entirely of binary digits (0-1)
L	Text contains only lowercase letters (a-z)
M	Text contains only letters (a-z, A-Z)
N	Text is a valid REXX number
S	Text is a valid symbol in REXX (e.g. a variable name)
U	Text contains only uppercase letters (A-Z)
W	Text contains an integer
X	Text contains only hexadecimal digits (a-z, A-Z, 0-9) and blanks can stand between hex pairs. Null string is also a hexadecimal character.

Example:

The following program lines come from a program that performs date conversions of the format YYYY/MM/DD in the format YYYY/DDD (Julian date) and vice versa. Before in the program any arithmetic operations are be

```
24  parse var indat yyyy"/"mm"/"dd .
25  if (datatype(yyyy,"N") * datatype(mm,"N") *,
26      datatype(dd,"N") = 0)  then return("error")
27  if mm > 12 | dd > 31 !,
28        (yyyy * mm * dd = 0) then return("error")
```

The PARSE command in line 24 parses the number entered in the variables YYYY, MM and DD. The IF statement in line 25 checks whether all three values are numeric. It works like this:

The DATATYPE function returns for all tests in which a correct number is found 1, otherwise 0. If only one of the tests provides 0, then the multiplication of all three tests is 0. This means that at least one value is not numeric.

Caution trap:

If anyone thinks now "You can do this much better by contracting the two IF statements in lines 25 and 27 into one IF statement." Then I can only say, "You have traded with lemons!" Why does this not work properly? This does not lead until then to an error if the program is actually called with wrong input values. Because then the following happens: All arithmetic operations of IF statements are executed to determine the result. This means in our case that the multiply operation yyyy*mm*dd is executed to complete the IF statement. Therefore, if one of the values contains a nonnumeric character, the program will cancel. Therefore, if someone should resist the temptation to contract these two IFs in one thing, he has built a classic time bomb here. The issue works until an incorrect value actually appears.

So, remember the following rule:

Rule:

Before you perform any arithmetic operations in a statement, you must previously perform tests in which no arithmetic operations occur, to ensure that all involved values are numerical!

By the way, what is the most embarrassing error in the programming?

The most embarrassing programming error:

Suppose you installed an extensive plausibility test in your program and this breaks in the reasonableness test without reporting the user's error. Then it is a bad thing for the user, because he does not know what he did wrong.

4.4 CONVERSION FUNCTIONS

With these functions, you can convert texts and numbers in other internal representations. Normally, you do not really need these features since in the TSO/REXX environment everything is very simple: All internal data representations are essentially stored in EBCDIC format. This means that:

– The number 1234567 is internally stored in seven bytes, with the following hexadecimal representation: "F1 F2 F3 F4 F5 F6 F7" X. In general, this representation is called the unpacked format.

– The text "I am happy!" has internal a length of 11 bytes and consists of the following hexadecimal characters: " C9 40 81 94 40 88 81 97 97 A8 5A" X.

The conversion functions are always very useful when values have to be converted into other forms of representation.

Example:

This small REXX program determines the job name directly from the system tables of z/OS.

Screen 4.1: Read the job name from CVT

```
/* DOC: REXX JOBNTSO                                          */
/* DOC: Determine the job name using the CVT of z/OS          */
/* DOC: CVT = Common Vector Table                             */
/* © FRANZ LANZ 2015                                          */
/************************************************************/
CVT  = STORAGE(10,4)                /* PLCCVT-PSA DATA AREA */
TCBP = STORAGE(D2X(C2D(CVT)),4)     /* CVTTCBP */
TCB  = STORAGE(D2X(C2D(TCBP)+4),4) /* TCB      */
TIOT = STORAGE(D2X(C2D(TCB)+12),4)   /* TCBTIO */
SAY STRIP(STORAGE(D2X(C2D(TIOT)),8)) /* TIOCNJOB */
EXIT
```

If you have the necessary knowledge of the z/OS operating system, try to decipher the REXX commands. Find the necessary knowledge about the conversion instructions in the following subsections. Of course, here all people who have ever programmed in assembler have an advantage. This applies to all conversion functions.

4.4.1 C2D – Character to decimal

This function returns the decimal value of a text. By specifying a length, negative numbers can be converted.

Format

number = C2D(text,[n])

text	This text is as binary text considered whose decimal value returned is.
n	If specified, this number indicates how many bytes are converted from text from the right. Is the first (i.e. the leftmost) bit of that number 1, then the number is output as a negative number in the two's complement.

Advice:

When using this function, it is important to ensure that NUMERIC DIGITS are set large enough. Otherwise, the call to C2D ends with an error.

In the following small program, I have compiled several calls to the C2D function. I think that you can understand the operation of C2D well.

Program 4.2: Program to show C2D functionality

```
/* DOC: REXX C2D     ***************************************************/
/* DOC: Examples of convert function C2D                               */
/* © FRANZ LANZ 2015                                                   */
/*********************************************************************/
numeric digits 22
say right(c2d("00000001"x),22)   /* A number one                      */
say right(c2d("7fffffff"x),22)   /* Largest positive number in word   */
say right(c2d("7fffffff"x,4),22) /* Largest positive number in word   */
say right(c2d("80000000"x,4),22) /* Largest negative number in word   */
say right(c2d("7fffffff"x,1),22) /* Only the rightmost byte           */
say right(c2d("7fffffff"x,2),22) /* Only the rightmost 2 bytes        */
say right(c2d("00007fff"x,2),22) /* Largest positive nbr in halfword  */
say right(c2d("00008000"x,2),22) /* Largest negative nbr in halfword  */
say right(c2d("0000007f"x,1),22) /* Largest positive number in bytes  */
say right(c2d("00000080"x,1),22) /* Largest negative number in bytes  */
say right(c2d("000000ff"x  ),22) /* Largest nbr in bytes without sign*/
say right(c2d("12345678"   ),22) /* Binary value of EBCDIC 12345678   */
say right(c2d("ABCDEFGH"   ),22) /* Binary value of EBCDIC ABCDEFGH   */
```

I have run the program. See the results below:

```
c2d("00000001"x)  →                                1
c2d("7fffffff"x)→                         2147483647
c2d("7fffffff"x,4) →                      2147483647
c2d("80000000"x,4) →                     -2147483648
c2d("7fffffff"x,1) →                              -1
c2d("7fffffff"x,2) →                              -1
c2d("00007fff"x,2) →                           32767
c2d("00008000"x,2) →                          -32768
c2d("0000007f"x,1) →                             127
c2d("00000080"x,1) →                            -128
c2d("000000ff"x  ) →                             255
c2d("12345678"   ) →    1743426534092878437 6
c2d("ABCDEFGH"   ) →    13961937044701104072
```

4.4.2 C2X – Character to Hexadecimal

Function:

This function returns the entered text in its hexadecimal notation. The output is then twice as long as the text entered.

Format:

number = C2X(text)

text	This text is an EBCDIC string, which is returned in hexadecimal representation.

Examples:

```
say c2x("1234567890") /* is in hex:F1F2F3F4F5F6F7F8F9F0*/
say c2x("ABCDEFGHIJ") /* is in hex:C1C2C3C4C5C6C7C8C9D1*/
say c2x("abcdefghij") /* is in hex:8182838485868788899 1*/
say c2x("!'$%&/()=") /* is in hex:4F7D7C5B6C50614D5D7E*/
```

4.4.3 D2C – Decimal to Character

Function:

With this function, you can convert a decimal number to the EBCDIC character format.

Format:

chars = D2C(number,[n])

n	If specified, this number indicates how many bytes are returned after the conversion of number. If n is greater than the number has digits, then is left-padded with spaces. If in converting, more characters emerge as defined by n, characters are truncated on the left.

Example:

```
numeric digits 22
say d2c(240,10)                  /* result:          0 */
say d2c(249,10)                  /* result:          9 */
say d2c(1396193704470l104072,10) /* result:   ABCDEFGH */
say d2c(17434265340928784376,10) /* result:   12345678 */
```

4.4.4 D2X – Decimal to Hexadecimal

Function:

With this function, you can convert a decimal number to its hexadecimal format.

Format:

hexchar = D2X(number,[n])

number	This is a decimal number whose content is returned in hexadecimal notation.
n	If specified, this number indicates how many bytes are returned after the conversion of number. If n is greater than the output length resulting from the conversion, the result is left-padded with 00 if it is a positive number, it is filled with hex FF, if it is a negative number. If in converting, more characters emerge as defined by n, left standing characters are truncated.

Examples:

```
numeric digits 22
say d2x(240,20)                  /* result: 000000000000000000F0     */
say d2x(249,20)                  /* result: 000000000000000000F9     */
say d2x(-15,20)                  /* result: FFFFFFFFFFFFFFFFFFF1     */
say d2x(1396193704470l104072,20) /* result: 0000C1C2C3C4C5C6C7C8     */
say d2x(17434265340928784376,20) /* result: 0000F1F2F3F4F5F6F7F8     */
```

Due to specifying the length of 20, the shaded places were prefixed on the left

4.4.5 X2B – Hexadecimal to Binary

Function

Convert a hexadecimal number into a binary string in text format.

Format:

bintext = X2B(hexnumber)

hexnumber	The hexadecimal string is converted into a binary string. The output is four times as long as the input.

Examples:

```
say x2b("ff")    /* result: 11111111                */
say x2b("11")    /* result: 00010001                */
say x2b("8AB1")  /* result: 1000101010110001        */
```

4.4.6 X2C – Hexadecimal to Character

Function

Convert a hexadecimal number to a string.

Format:

text = X2C(hexnumber)

hexnumber	This hexadecimal number is converted to its decimal representation.

Examples:

```
say x2c("f0f1f2f3f4f5f6f7f8f9") /* result: 0123456789 */
say x2c("c1c2c3c4c5c6c7c8c9d1") /* result: ABCDEFGHIJ */
say x2c("8182838485868788991") /* result: abcdefghij */
say x2c("4F7D7C5B6C50614D5D7E") /* result: !'$$%&/()= */
```

4.4.7 X2D – Hexadecimal to Decimal

Function

Convert a hexadecimal number to decimal.

Format:

decnum = X2D(hexnumber,[n])

hexnumber	This hexadecimal number is to its decimal representation converted.
n	If entered, this number indicates how many bytes are converted from hexadecimal value from the right. When the first (i.e. the left-most) bit of number is 1, then the number is returned as a negative number in the two's complement. If this number is zero, then the returned value will be a positive number.

Note:

When using this function, it is important to ensure that NUMERIC DIGITS are set large enough. Otherwise, the call to X2D ends with an error.

Program 4.3: Program to show X2D functionality

```
/* DOC: REXX X2D      ************************************************/
/* DOC: Examples of convert function X2D                            */
/* © FRANZ LANZ 2015                                                */
/********************************************************************/
numeric digits 22
say right(x2d("00000001"),22)  /* a one*/
say right(x2d("7fffffff"),22)  /* Largest positive nbr in the fullword*/
say right(x2d("7fffffff",4),22)/* Largest negative nbr in the fullword*/
say right(x2d("80000000",4),22)/* Largest positive nbr in the fullword*/
say right(x2d("7fffffff",1),22)/* Only the rightmost byte           */
say right(x2d("7fffffff",2),22)/* Only the rightmost 2 bytes        */
say right(x2d("00007fff",2),22)/* Largest positive number in halfword */
say right(x2d("00008000",2),22)/* Largest negative number in halfword */
say right(x2d("0000007f",1),22)/* Largest positive number in byte    */
say right(x2d("00000080",1),22)/* Largest negative number in byte    */
say right(x2d("000000ff"  ),22)/* Largest number in bytes without sign.*/
```

I have run the program. See the result here:

```
x2d("00000001")   →                    1
x2d("7fffffff")   →           2147483647
x2d("7fffffff",4) →           2147483647
x2d("80000000",4) →          -2147483648
x2d("7fffffff",1) →                   -1
x2d("7fffffff",2) →                   -1
x2d("00007fff",2) →                32767
x2d("00008000",2) →               -32768
x2d("0000007f",1) →                  127
x2d("00000080",1) →                 -128
x2d("000000ff"  ) →                  255
```

4.5 FORMATTING FUNCTIONS

These functions formatting data for output.

4.5.1 CENTER – Centering a string

Function

With this function, a string can be arranged so that the string is in the middle and on the left and right with the pad character padded.

Format:

result = CENTER(intext,length,[pad])

intext	Input text, extended or shortened as defined by the length parameter.
length	Number of characters of the result text. Is length less than the length of intext, then the input text is equally shortened on both sides.
pad	Here, a fill character to specify which of the left and right of intext is added for an enlargement. Standard for the pad character is a blank.

4.5.2 COPIES – Reproduce texts

Function:

With this function a string can be duplicated n times.

Format:

text = COPIES(intext,n)

intext	Input text that is n times duplicated.
n	Number of copies.

Example:

```
say copies("**",10) → **********
```

4.5.3 FORMAT – Format numbers

Format numbers for output, as you like.

Format:

Text = FORMAT(number,before,after,expp,expt)

Table 4.5: Description of parameters for the FORMAT function

Parameter	Meaning
number	Expression or number, which will be formatted.
before	Number of digits before the decimal point.
after	Number of digits after the decimal point.
expp	Number of digits in the exponent.
expt	Limit of digits of number when the output is automatically switched to exponential representation.

4.5.4 JUSTIFY – Formatting a string

Function

Block formatting of a string consisting of words by adding padding characters between the words.

Format:

text = JUSTIFY(intext,length[,pad])

intext	Input text that normally consists of words, each separated by one or more blanks.
length	Length of the result text.
pad	Character to be inserted between the words. Default is a blank.

Example:

```
say justify("This  is a  Text",40,"-")
say justify("Word              ",40,"-")
```

These commands give the following texts:

This---------is--------a----------Text
Wort----------------------------------

4.5.5 LEFT – Rearrange text left-justified

Function

Left-align the input text and pad it on the right side with padding character.

text = LEFT(intext,length[,pad])

intext	Input text.
length	Length of the resulting text.
pad	Padding character. Default is the blank.

4.5.6 RIGHT – Arrange text right- justified

Function

Right-align the input text and pad it on the left side with padding character. Default padding character is blank.

Format:

text = RIGHT(intext,length[,pad])

intext	Input text.
length	Length of the resulting text.
pad	Padding character. Default is the blank.

4.6 STRING FUNCTIONS

The string functions form a powerful group of agents in the REXX programming. Except for a CHANGE function, I have yet to miss any string function. Therefore, I have written a long time ago a CHANGE function using REXX. The REXX source code of this CHANGE function is in the SMART ISPF utilities. The CHANGE function is described in section 4.6.3 SCHANGE – Change of texts on page 86.

4.6.1 DELSTR – Delete substrings

Function:

With this function, you can remove a portion of a text. The text will shortened by the deleted part.

Format:

text = DELSTR(intext,start[,number])

intext	Input text.
start	Starting position of the text to be deleted in intext. When the starting position is greater than the length of the intext then intext is returned unchanged.
number	Number of characters, which are deleted. The text will be deleted up to the end when this parameter is not specified.

4.6.2 INSERT – Insert text

Function:

With INSERT, you can insert a text in another text. This is in a sense the inverse function of DELSTR.

Format:

text = INSERT(intext,text[,start[|,length][|,pad])

intext	Inserted text.
text	Text to which intext is inserted.
start	Start position in text after the insertion is to occur. When the starting position is omitted or zero, then intext is inserted at the beginning of text.

	this parameter is not specified then the current length of intext is automatically used.
pad	If the entered length is greater than the current length of intext then this character expands intext on the right as defined in length.

Example

See next function CHANGE.

4.6.3 SCHANGE – Change of texts

This function is not part of TSO/E REXX. Rather more, I developed it because a change function in the daily work with REXX in the TSO/ISPF environment in my opinion is essential. This function is part of the SMART ISPF utilities. See section 15.5.12 REXX subroutine SCHANGE – REXX change function on page 283.

Function:

Change of texts.

Format:

text = SCHANGE(intext,searchtext[,replacement][,testtext])

intext	Text to be changed.
searchtext	Search text. Change replaces all texts that are found with searchtext with the replacement text in a single call.
replacement	This text should replace the search text. When the replacement text is omitted or contains a null string, then searchtext is only removed from intext.
testtext	The CHANGE shall only take place if this text is included in the intext. Of course, it does not make sense when searchtext and testtext are equal. However, the function still runs properly.

4.6.4 LENGTH – Length of a text

Function

With this function, the length of a text is determined.

Format:

n = LENGTH(text)

N is the length of a text. If a text is a null string, then is n = 0.

4.6.5 OVERLAY – Superimpose text

Function:

With OVERLAY, you can insert a text into another text after the specified location.

Format:

text = OVERLAY(intext,text[,start][,length][,pad])

intext	Text inserted by overlay.
text	Text to overlie beginning at start.
start	Start position in text from which the insertion is to occur. When the starting position is missing, 1 is assumed. When the parameter start is specified, it must contain a positive integer.
length	Number of characters to be overlaid. When this parameter is not specified, the current length of intext will automatically be used.

	overlay is performed.

4.6.6 POS – Search for text

Function:

Search for a text within another text. The starting position of the found text within the input text will be returned. This function provides the same functionality as the INDEX function. According to literature, however, you should use POS.

Format:

```
n = POS(needle,haystack[,start])
```

needle	This text is searched for in the haystack
haystack	Scanned text.
start	From this position in the haystack, the search is started.

4.6.7 STRIP – remove border characters

Function:

When passing parameters and texts it often happens that these texts are featured at the front and at the rear with blanks or other characters which must be removed. This function removes leading and trailing (same) characters from a text. All same leading and trailing characters will removed. Since many operations, especially when passing parameters with leading and trailing blanks, STRIP is often used to remove these blanks. This function is often used just for precaution.

Format:

```
text = STRIP(intext[,option][,char])
```

intext	This text will be stripped.
option	This option specifies whether only leading, trailing, or characters from both ends should be removed from text. **B**oth Removes the characters at both ends. This is the default value. **L**eading Removes leading characters only. **T**railing Removes trailing characters only.
char	These are the removed characters. Default is blank.

1 Tip:

The STRIP function is often required in order to free DSNs from the surrounding single quotes ('). Sometimes it happens that a DSN is entered once with and once without the quotes. It often happens that you have to insert the DSN in a function that absolutely needs quotation marks. In such cases, I use the following statement:

```
dsn1 = translate(dsn,"    ",".'()")
```

What does this statement?
First, any existing single quotes will be removed. Thereafter, any existing blanks will be removed and finally the DSN will set back in quotes.
Now it does not matter in which form the DSNs parameters are passed. With or without quotation marks and blanks. The result is always a DSN with quotation marks and without blanks.

4.6.8 SUBSTR – Extract part of a text

Function

This function extracts a text part of a text.

Format:

```
teil = SUBSTR(text,n[,length][,pad])
```

text	Text from which the substring is taken.

length	Number of characters, which are extracted. When length is not specified, all characters will be taken up to the end of text.
pad	If length is greater than the available text, then the resulting text is extended with the padding character. The default pad is the blank.

4.6.9 TRANSLATE – translate characters

Function:

This function translates individual characters of a text to other characters. Translation is done in such a way that the same characters in the translation table replace the characters found in the search table.

Format:

text = translate(intext[,replacetab][,searchtab][,pad])

intext	This is the text that will be translated. If only intext is specified,then this is translated to uppercase.
replacetab	This text contains characters that are replaced in intext if these characters are found in the search table.
searchtab	This text contains characters sought in intext.
pad	The fill character replaces such positions in intext where characters were found using the search table, but do not have a corresponding position in the replace table.

Examples:

```
orgdsn = strip(translate(orgdsn," ","'"))
```

This command first replaces all apostrophes in orgdsn by blanks and then removes all surrounding blanks from orgdsn.

argl = translate(arg(1))

With this command, the first argument that was passed by a calling program is set to uppercase.

```
dsn1 = translate(dsn,"    ",".'()")
```

This command replaces the characters dot, apostrophe, left and right brackets by blanks. The resulting string contains all qualifiers of the DSN and an eventually contained member name as single words. To get the member name, the following command can be used:

meml = word(dsn1,words(dsn1))

4.6.10 VERIFY – verify text

Function:

VERIFY checks whether a text only consists of characters of a second text or not. This function is very good to use for validation of texts such as DSNs.

Format:

n = VERIFY(text,checktext[,option][,start])

text	This is the text which is to be checked.
checktext	These characters are sought in TEXT.
option	Here you can specify how to perform the check:

Nomatch	When specifying N, the function returns zero when ALL of the characters of text are also included in check text. Otherwise, the position of

Match	When specifying M, the position of the first character in the text, which is also included in test text will back given. If no character from check text is found in text, then zero will be given back.

start	Start position in text where the search begins.

4.7 WORD FUNCTIONS

One of the greatest strengths of REXX are the word functions. This is therefore so useful because just in the z/OS world very many terms appear in ISPF, programs, JCL, etc., as by blanks or special characters separated texts. From a text such as a DSN, you can very quickly use TRANSLATE to remove the special characters and you have a text that consists only of words.

Definition of a word in REXX sense:

- A word is a coherent text that does not contain blanks.
- Words in a string are separated from each other by one or more blanks.

4.7.1 WORD – return of a word

Function

WORD returns the nth word back from a string.

Format

ywort = WORD(intext,position)

intext	String from which the addressed word is extracted by position.
position	Position number of word to be returned. If position is greater than the number of words in intext then a null-string will be returned.

4.7.2 WORDINDEX – return the starting position of a word

Function:

WORDINDEX returns the position of the first character of the nth word of a string.

Format:

pos = WORDINDEX(intext,position)

intext	String in which the selected word is addressed by position.
position	Number of the word within intext, of which the position of the first character is returned.

Examples are at the end of the section Word Functions found.

4.7.3 WORDLENGTH – return the length of a word

Function:

WORDLENGTH returns the length of the nth word in a string.

Format:

length = WORDLENGTH(intext,position)

intext	String in which the by position selected word is addressed.
position	Number of the word whose length will be returned.

WORDPOS searches a word in a string and returns its position.

Format

pos = WORDPOS(word,string[,start])

word	Searched word. If the word is not found in the string, zero will be returned
string	String composed of several words.
start	Word position where search starts.

4.7.5 WORDS – number of words in a string

Function

WORDS returns the number of words contained in a string.

Format:

number = WORDS(intext)

intext	String whose number of contained words are to be given back.

Program 4.4: WORDFUNC – Examples of WORD function

```
/* DOC: WORDFUNC REXX MAIN                                              */
/* DOC: Examples of the use of the word functions                       */
/*---------------------------------------------------------------------*/
/* © FRANZ LANZ 2015                                                   */
/***********************************************************************/
dsn = "'PROX.SMART.REXX'"
dsw = translate(dsn,"  ",".'") /* Remove dots from DSN*/
say word(dsw,1)" = HLQ"
say word(dsw,words(dsw))" = LLQ"
say wordpos("SMART",dsw)" = Position of the word SMART"
say wordlength(dsw,3)" = Length of the 3rd word"
```

This program produces the following list:

```
PROX = HLQ
REXX = LLQ
2 = Position of the word SMART
4 = Length of the 3rd word
```

4.8 SYSTEM FUNCTIONS

The system functions return information about the z/OS system and its resources under which control a REXX program is currently running. These functions are described in the literature as TSO/E External Functions. They can be found under this heading in the brochure TSO/E REXX Reference.

4.8.1 LISTDSI – List data set information

Function:

This function returns a wealth of information concerning a data set. A detailed description of this function is found in section 4.8.1 LISTDSI – List data set information on page 93. If you need any information about a file, you must call up three functions. These are the functions LISTDSI, DSINFO and LMDLIST. Practical examples of all three functions can be found in the source code of the SMART ISPF utility SSC, in the section 15.4.23 Program SSS – Perform a Super-Search on page 271. See also section 8.2.2 DSINFO – ISPF service which provides data set information on page 162 and 8.2.1 LMDLIST – Data set list service on page 161.

4.8.2 MSG – control of the TSO messages

Function:

With MSG, you can specify whether the possibly during the program run occurring TSO messages are outputted directly or not. This is sometimes very helpful when you get TSO messages at a point in the program where you would like to suppress them.

Format:

msgstat = MSG("ON"|"OFF")

This call returns the status of MSG, which can be ON or OFF. This function is very useful when you wonder why TSO messages do not appear.

x = msg("ON")	This call switches the display of the TSO messages to ON.

```
msg("OFF")
```

Example:

```
x = msg("OFF")
address "TSO" "delete 'ldsn'
x = msg("ON")
```

Note:

The TSO DELETE command always produces a message. In this case, the active ISPF panel disappears and the delete message will be displayed. Since this is very annoying and I do not want to see the message from TSO DELETE, here I suppress the output of the message by setting x = msg("OFF").

4.8.3 MVSVAR – Return z/OS system information

MVSVAR returns various information about the z/OS system. A parameter keyword is mandatory defining the value you wish to receive. I have often used only one of these values. This is the name of the z/OS LPAR. That is the four-character logical partition name where my program is running. You get this value as follows:

lpar = MVSVAR(SYSNAME)

If you need more information about this feature, view the detailed description of this function in the brochure **TSO/E REXX Reference**.

4.8.4 OUTTRAP – take TSO messages

Function:

OUTTRAP is a very frequently used function. REXX programs use OUTTRAP to store TSO outputted messages in a **stem**. You can use these TSO messages in your REXX as you like.

Format:

x = outtrap(name.)

This call ensures that from now on all TSO messages are written in the stem **name**.

x = outtrap("OFF")

This call finishes writing of TSO messages in the stem name. TSO stores the number of rows of the stem in **name.0**.

Example 1: Intercepting the messages of the SUBMIT command.

```
1 x = outtrap(text.)
2 address TSO "SUBMIT * END($$)"
3 x = outtrap("OFF")
4 do i = 2 to text.0; say text.i; end i;
```

Explanation of the REXX statements:

Lines	Explanation
1	With this statement, the TSO is informed that from now messages are no longer sent to SYSTSPRT, but instead written into the stem text.
2	TSO executes the SUBMIT command and writes the resulting messages in the stem text.
3	The trapping of the TSO messages will be switched off. The variable text.0 now contains the number of the TSO message lines in the stem text.
4	The messages from row two will be outputted. Since SUBMIT usually produces only two lines and the text that interests us is always in the second line, we only print this line. The output line might be: **JOB LANZT373(JOB06586) SUBMITTED**

Example 2: Outputting the file information and members list of a PDS.

Program 4.5: Example of calling the TSO function LISTDS

```
/* DOC: REXX LISTDS                                  */
/* DOC: Example of calling the TSO function LISTDS*/
/* © FRANZ LANZ 2015                                 */
/***************************************************/
X = outtrap("inmem.*")
"LISTDS 'PRTT.REXX' MEMBERS"
X = outtrap("OFF")
do i = 1 to inmem.0
   say inmem.i
end i
```

Printed output:

```
PRTT.REXX
--RECFM-LRECL-BLKSIZE-DSORG
  FB     80    27920    PO

--VOLUMES--
  D1D115
--MEMBERS--
  $CONCAT
  $LPARS
  BNDUMSPP
  EINSATZS
  EXNAME
  TT
```

Remark:

I will not describe the LISTDS function here in detail. The above example may suffice as information. The description of the TSO function LISTDS is contained in the brochure **TSO-E Command Reference**.

4.8.5 SYSDSN – check data set status

Function

Use SYSDSN to determine whether a file exists and that you can access it.

Format:

text = SYSDSN(DSN)

DSN must appoint a cataloged file.

The SYSDSN function returns ONE of the following texts, which are self-explanatory:

```
OK   /* data set is available and can be used */
MEMBER NOT FOUND
MEMBER SPECIFIED, BUT DATASET IS NOT PARTITIONED
DATASET NOT FOUND
ERROR PROCESSING REQUESTED DATASET
PROTECTED DATASET
VOLUME NOT ON SYSTEM
INVALID DATASET NAME, dsname
MISSING DATASET NAME
UNAVAILABLE DATASET
```

Example:

The following program part allocates a data set. If the data set already exists, it is only allocated. If it does not exist, it will be created.

```
address TSO
if sysdsn(ldsn) = "OK" then "alloc dd("ddname")",
                            "dsn("ldsn") old reuse"
else "alloc dd("ddname")",
     "dsn("ldsn") new reuse space(15) tracks",
     "recfm(v b a) lrecl(133)"
"execio "zl" diskw "ddname" (stem out. finis"
"free dd("ddname")"
```

4.8.6 SYSVAR – get system Information

Function

With SYSVAR, similar to MVSVAR, some system information can be retrieved. Here only two information points at run around procedures are of interest:

Example 1: Does a program run in foreground or in background.

foreground means: The program runs in the ISPF online.

background means: The program runs under ISPF that is started as a batch job.

Remark:

It is important to know this operating mode, because maybe files are be assigned in a batch job via DD statement in the JCL, while in foreground mode the TSO alloc command must be used. On the other hand, any error messages in background mode differently than in foreground mode are outputted. For the query where the program runs, you can use the following statement:

text = SYSVAR(SYSENV)

FORE → The REXX program runs in foreground.

BACK → The program runs in background in a batch job.

Example:

Program 4.6: Control messages display ONLINE and in BATCH

```
dll_exit:
arg rc text
zedlmsg = "Error in procedure >read_data_lmget<. RC="rc text
if zerrlm <> "ZERRLM" then zedlmsg = zedlmsg "ISPF-Error:"strip(zerrlm)
if sysvar("SYSENV") = "FORE" then "ISPEXEC SETMSG MSG(ISRE001)"
else do
   say exname zedlmsg
   zispfrc = 12 /* The batch job ends wit RC = 12 */
   "ISPEXEC VPUT (ZISPFRC) SHARED"
end
exit 12
```

If the procedure is running in the foreground, the error message is displayed in an online window using SETMSG. Otherwise, the error message is sent via SAY to the SYSTSPRT DD in the batch job.

Example 2: Does the program run in the ISPF environment runs

If you execute REXX procedures, which need ISPF services in batch jobs, then you should definitely check whether the ISPF is also available in the batch job. If not, you have to abort the procedure with an appropriate error message. If the procedure continues, there may be inexplicable crashes.

For this query, you can use SYSVAR(SYSISPF). This call returns ACTIVE when the procedure is running under ISPF. It returns NOT ACTIVE when the procedure only runs under TSO.

Example:

```
If sysvar(sysispf) <> "ACTIVE" then do
   rc = 20
   say "ISPF is not active, procedure ends"
   exit rc
end
```

4.8.7 STORAGE – read and write memory contents

Function:

With this function, you can read out memory contents and overwrite them. The possibility of overwriting should be handled with caution. This function is normally to read memory contents, which contain interesting information used.

Format:

value = STORAGE(address,length,text)

value	Result.
address	Specifies the memory address from which the data are collected in length. This address must be specified directly in the form of hexadecimal digits.
length	Specifies how many bytes are to be read or written at the starting address.
text	If anything is specified here, this text replaces the memory contents beginning with address in the length of text. Before replacing the contents starting at the addressed memory location, the previous contents are stored in the variable value.

Note:

The replacement of memory locations should be handled very carefully. You can shoot down your own address space. However, you do not need to worry, because with the security rights that you have as a normal user, you cannot override system areas.

Example 1:

The **Common Vector Table (CVT)** contains pointers to other tables in the z/OS system. Therefore, if you want to read data from these tables, such as the job name, then you first have to read the contents of the CVT . This address is common in all z/OS systems at the memory location 16. The number 16 expressed as a hexadecimal number is 10. The pointer to the CVT is always four bytes long.

The result of the command say c2x(storage(10,4)) is in my currently being

Starting from this address, you can now point through all control blocks until you have found the information you need. You can find an example in Screen 4.1 Read the job name from CVT on page 77.

Example 2

I found another example of the application of the STORAGE function on the Internet. It gets the name of the program that is currently executing.

Program 4.7: PROGNAME examples for using STORAGE function

```
/* REXX PROGNAME                                          */

/* Subject: How to find program executed under MVS        */
/* Author: Andy Wood                                      */
xp      = storage(0000021c,4)
xp2     = ptr(xp,'b4'x)
xp3     = d2x(c2d(xp2) + c2d('168'x))
envname = storage(xp3,8)
say 'jobstepprogname=' envname
exit
ptr: procedure
   address = c2d(arg(1))
   offset  = c2d(arg(2))
return storage(d2x(address+offset),4)
```

Note:

If this program is executed online under ISPF, the displayed name is always IKJEFT01. This is the TSO processor.

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
3 The REXX commands

NEXT
5 The TSO/E REXX commands