interskill learning

REXX Environment Functions

Objectives

REXX Environment Functions

In this module, you will explore the REXX environment functions that return settings defined by the system and the REXX program code itself.

After completing this module, you will be able to:

- Identify How to Use the Miscellaneous Environmental Functions
- Identify How to Use Error Condition Environmental Functions
- Identify How to Use System Environmental Functions

String functions	Perform various comparison, interrogation, and manipulation actions on data strings.
Text and word functions	Interrogate and manipulate words and specific data within a string.
Justification functions	Justify and format text and data strings.
Numeric functions	Interrogate and format numeric values.
Character conversion functions	Convert and manipulate binary, hex, and character values.
Environment functions	Interrogate the environment that the REXX program is running under, and return settings and definitions.
Stream I/O functions	Used for file processing on many platforms.

For the purposes of this course, we have divided the built-in functions into the groups listed above.

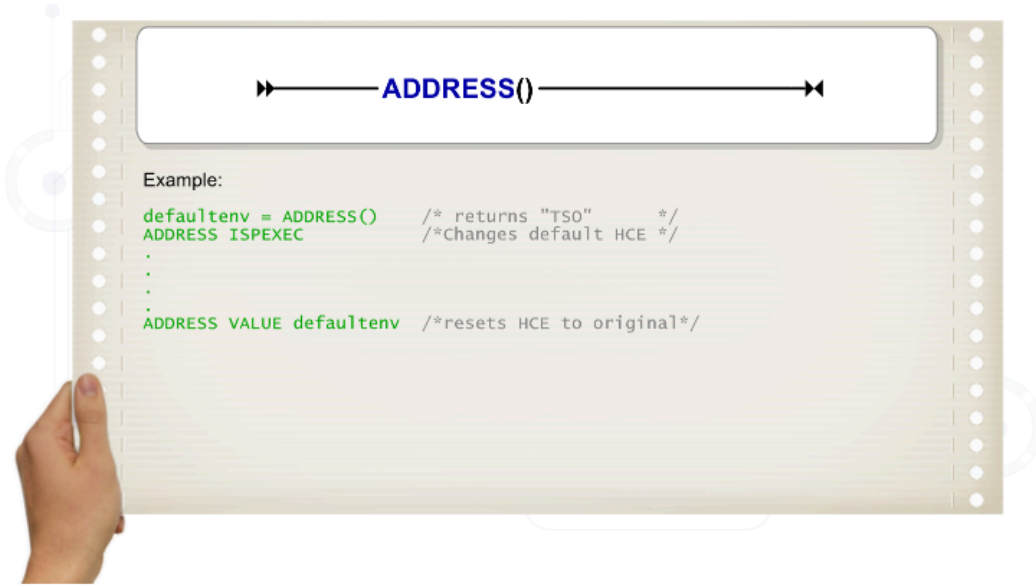
You will now focus on the environment functions, which return values related to the environment that the REXX program is running under, and values specifically related to the current execution of the program.

To help you understand the environment functions group, we have divided it into three subgroups: miscellaneous, error condition, and system environment functions.

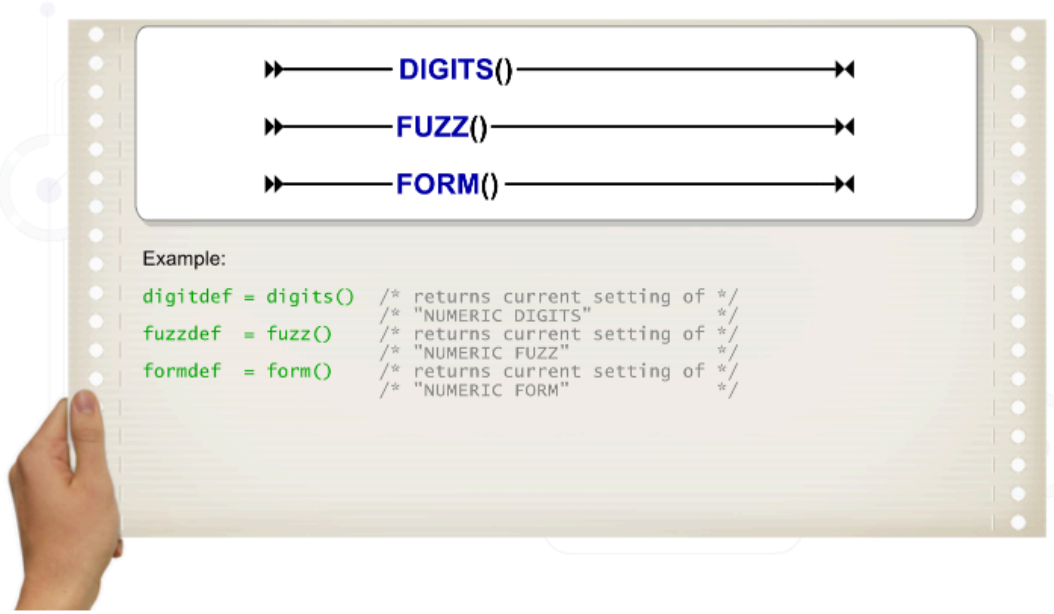
ADDRESS	Returns the name of the environment of the host machine to which commands are being submitted
DIGITS	Returns the current setting of NUMERIC DIGITS
FORM	Returns the current setting of NUMERIC FORM
FUZZ	Returns the current setting of NUMERIC FUZZ
QUEUED	Returns the number of lines remaining in the program stack or external data queue
VALUE	Returns the value of a specified REXX symbol; useful if a variable contains the name of another variable
TRACE	Returns the current value of the TRACE keyword instruction and optionally sets it to a new value
EXTERNALS	Returns the number of elements in the terminal input queue; always returns 0 in TSO/E
LINESIZE	Returns the current terminal line width minus 1; LINESIZE is 32 if REXX is running in TSO/E background

The first subgroup that you will look at is a miscellaneous group that interrogates several different areas and values used by the program.

The EXTERNALS and LINESIZE functions are not covered in this course and the TRACE function was reviewed earlier.

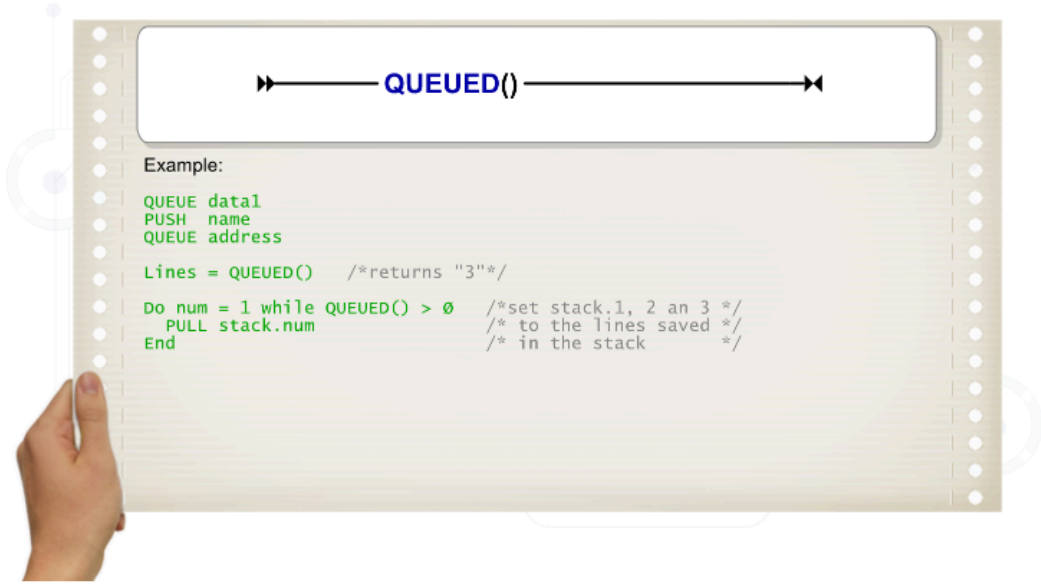


The ADDRESS function has no parameters and returns the current default host command environment (HCE). This function will save the current environment in a variable before changing it so it can be used later to return to the original HCE.



The DIGITS, FUZZ, and FORM functions return the current value set by the system or the NUMERIC keyword instruction for the DIGITS, FUZZ, and FORM parameters.

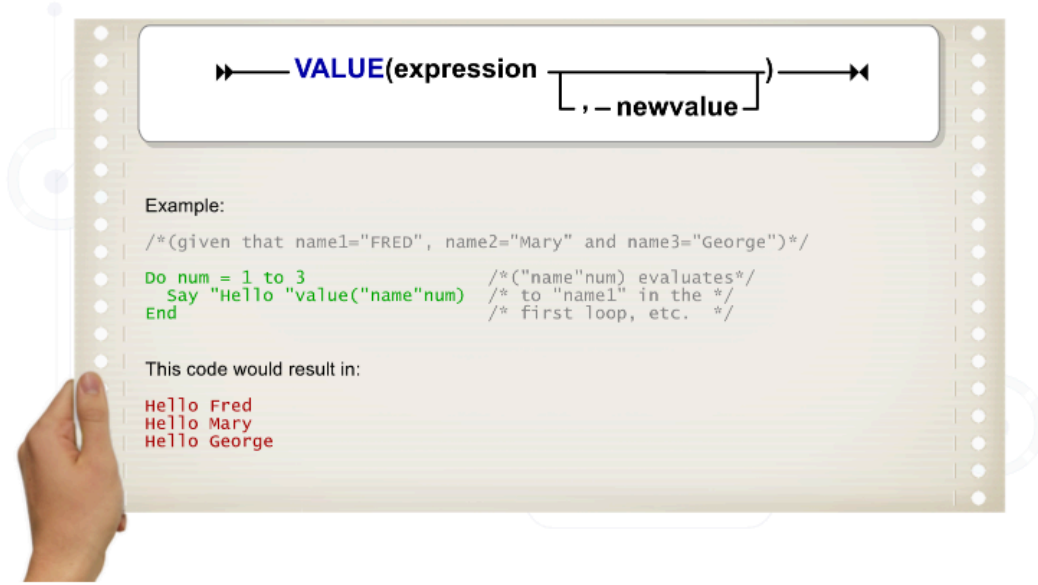
Like the ADDRESS function, these functions have no parameters but are used to save the current values before changes are made so they can be reset at a later stage.



The external data queue or stack is used to temporarily store records or data to be used by other subsystems or REXX routines in this address space. The QUEUED function can be used to determine the number of records stored in the stack.

The above example shows how a loop can be coded to PULL all the records from the stack by using the QUEUED function as a control value, and store them in a compound variable.

The QUEUED function has no parameters.



The VALUE function enables symbols or variable names to be expressions that are evaluated before symbolic substitution is performed. In effect, it enables a variable name to be a variable.

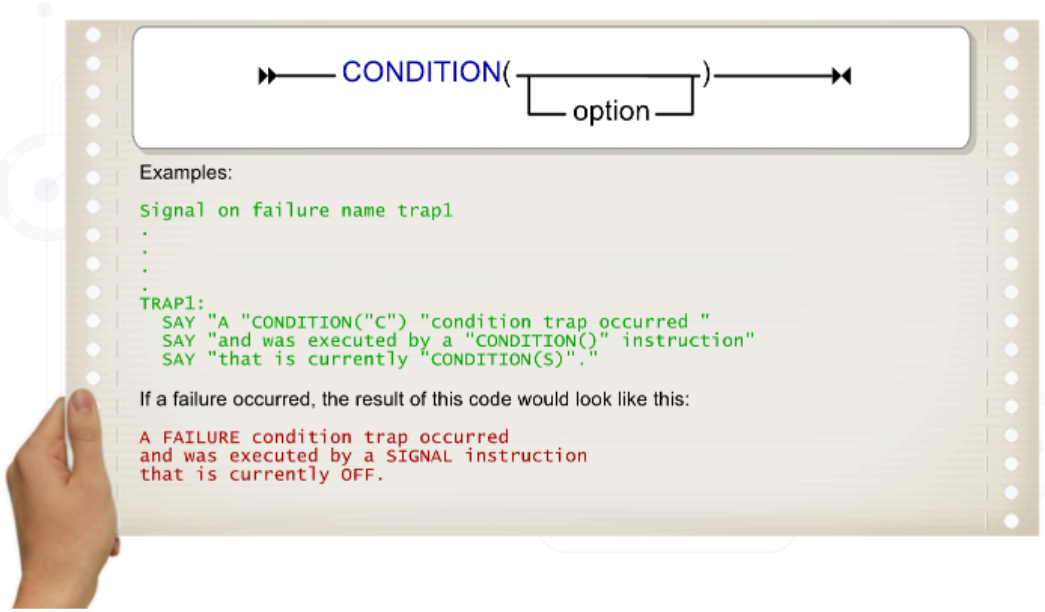
The expression passed as a parameter is evaluated and then the interpreter looks for a variable of that name and returns its value. Optionally, it can also reset the value to `new_value`.

Some environments, such as ISPF, cannot process compound variables. The above example shows how VALUE can be used for numbered simple variables.

CONDITION	Returns the condition information, such as condition name, description, instruction, and status associated with the current trapped condition
SOURCELINE	Returns the line number of the final line in a REXX program or the text of a specified line number in a REXX program
ERRORTEXT	Returns the error text associated with a given error number

The SIGNAL ON condition or CALL ON condition keyword instructions can be used to define error condition traps with the SIGL system variable containing the line number of the code that triggered the trap.

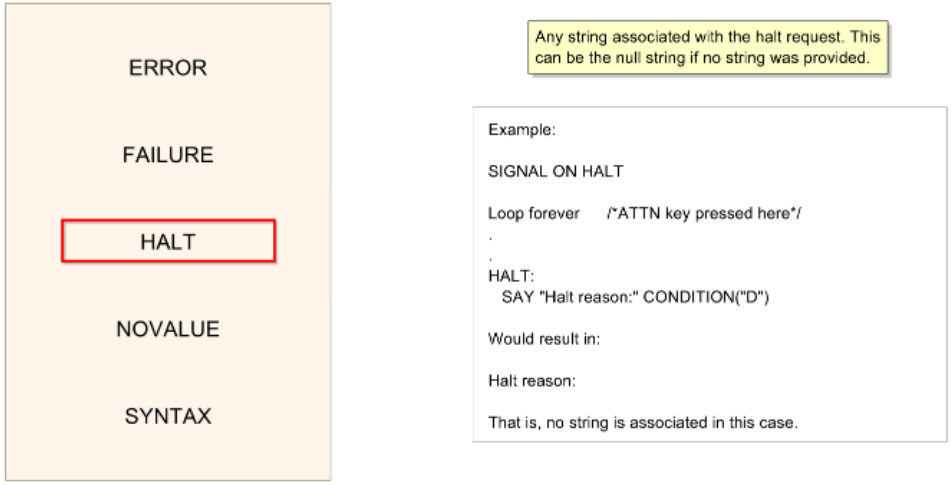
The CONDITION, ERRORTEXT, and SOURCELINE functions are used to analyze information about the error and inform the programmer.



The CONDITION function describes how the condition trap was triggered with CALL or SIGNAL, and the cause of the trap being triggered. The options that can be coded are:

Condition - Returns the name of the condition trap: ERROR, FAILURE, HALT, NOVALUE, or SYNTAX
Instruction - This is the default; returns the instruction that set the trap condition: CALL or SIGNAL
Status - Returns the current status of the trap that was triggered: ON, OFF, or DELAY
Description - Returns a character string describing the current trapped condition

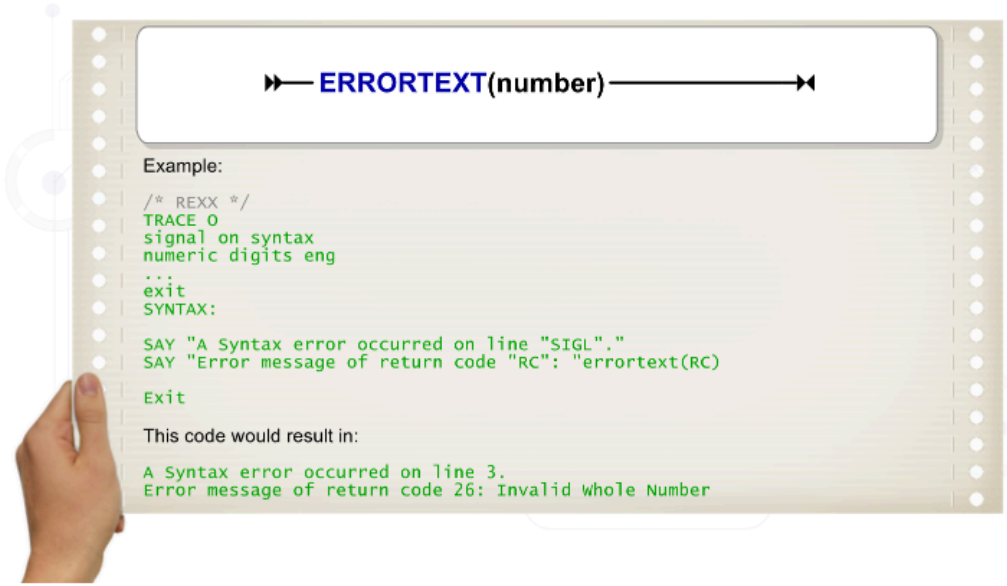
If coded, only the first character of the option is required. All other characters are ignored.



When using the Description option of the CONDITION function, the result is a description string related to the current triggered trap.

The format of this string varies depending on the type of trap triggered. For example, if the trap is defined by a NOVALUE condition, the descriptive string will contain the variable name that is detected with no value.

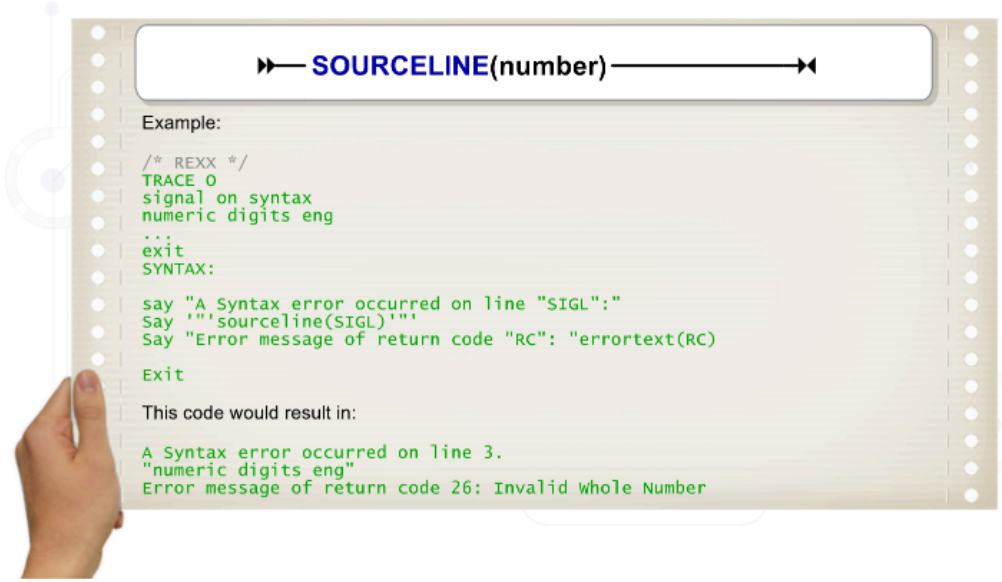
Mouse-over each trap condition to see the value of the CONDITION("D") descriptive string.



When a syntax error occurs in REXX processing, the message may not appear on the screen under certain circumstances, or it may be necessary to interrogate the message to determine the recovery process that is required.

Syntax errors all produce a return code that is saved in the system variable RC. The ERRORTEXT function can be used to save or display the error message associated with a particular syntax return code.

Shown here is the syntax of the ERRORTEXT function.



Although CONDITION(D) often provides useful information, it does not always indicate the actual line of code from which the condition trap was triggered.

The SOURCELINE function returns the line of code from a specific line in the program. As the system variable SIGL contains the line number of the code that triggered a condition trap, it is possible to interrogate and display the problem code.

Shown here is the syntax of the SOURCELINE function.

```
/* REXX */
signal on syntax name trap1
CALL ON ERROR name trap1
Signal on failure name trap1
Signal on novalue name trap1
.
.
exit
Trap1:

say "A "condition("C") occurred on line "SIGL":"
Say '"'sourceline(SIGL)'"'
SELECT
 when CONDITION("C") = "SYNTAX" then
  do
   Say "Syntax error '"CONDITION(D)"' occurred with "
   Say "return code "RC". Reason: "errortext(RC)
  end
 when CONDITION("C") = "ERROR" |,
      CONDITION("C") = "FAILURE" then
   Say "Command Error return code: "RC"."

when CONDITION("C") = "NOVALUE" then
   Say "Variable "CONDITION(D) has not been set."

 otherwise nop
END
If condition() = "CALL" and RC < 8 then return

Do while queued() > 0
 Pull
End

exit
```

For NOVALUE conditions, display the uninitialized variable.

For example:

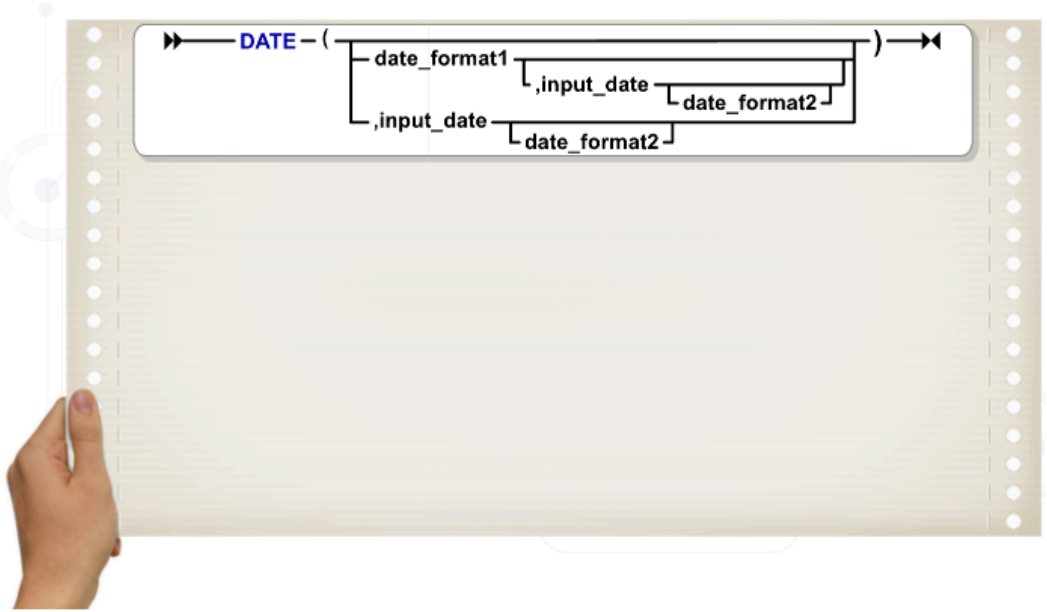
```
A NOVALUE trap occurred on line 45:
"SAY Hello name"
Variable HELLO has not been set.
```

These functions help diagnose and debug problems when coding error routines. The above code could be coded to set up a generic error routine by using several of the facilities and functions you have just explored.

Mouse-over the code for a description of each of the clauses.

DATE	Returns the system date in one of a variety of optional formats
TIME	Returns the system time in one of a variety of optional formats
USERID	Returns the name of the TSO/E userid: z/OS platform only

The system environmental functions listed above interrogate values set by the system or platform that the REXX program is running on.

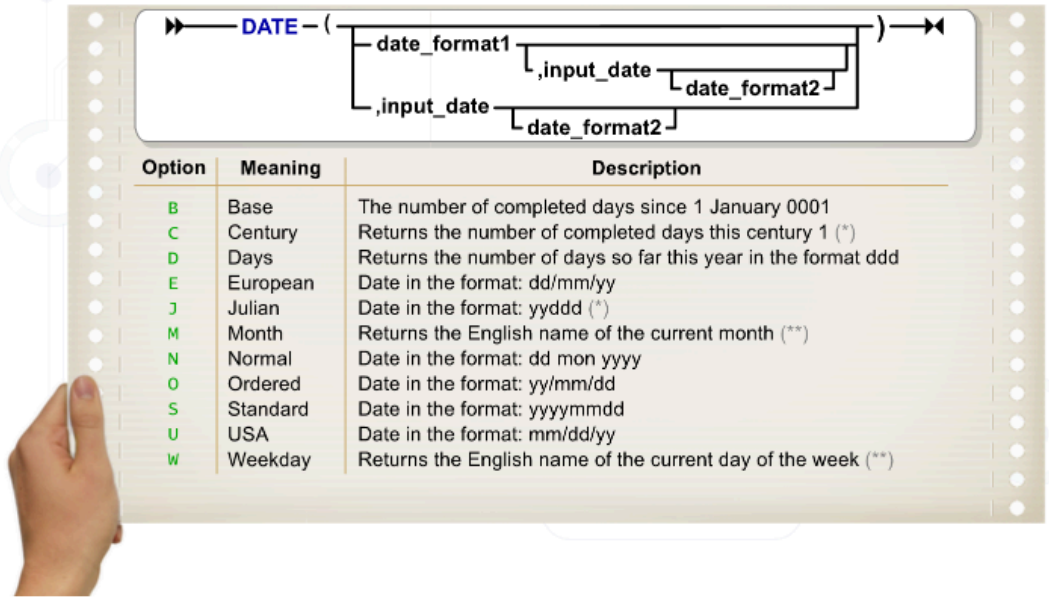


The `DATE` function can return a date based value from `input_date`. If `input_date` is not specified, the default is the current date.

If `date_format1` or `date_format2` are not specified, the default option is `N`ormal (dd mon yyy); `date_format1` is used to specify the output format of a date.

`date_format2` is used to define to REXX the current format of `input_date`.

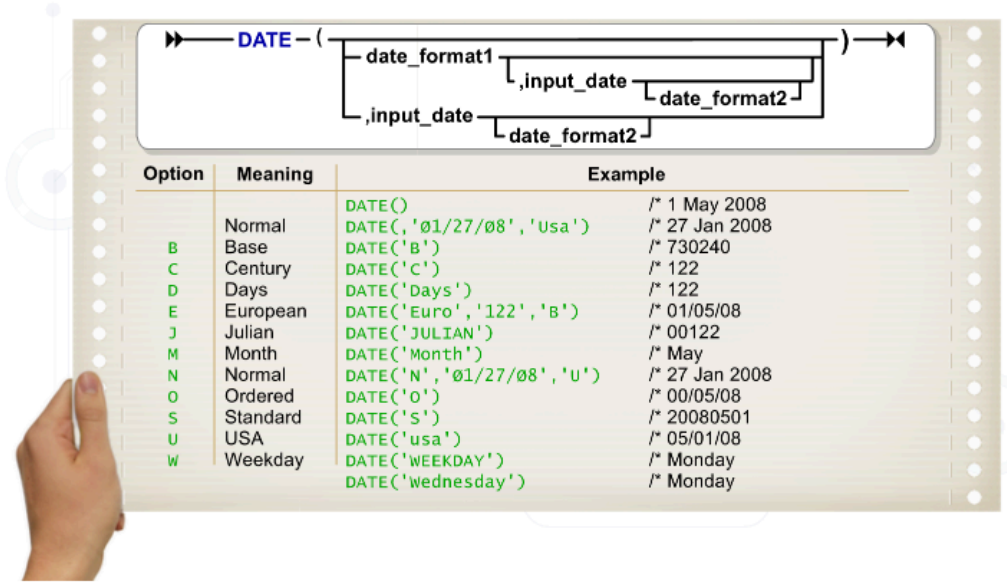
`input_date` is a date to be converted to `date_format1` from `date_format2`.



Option	Meaning	Description
B	Base	The number of completed days since 1 January 0001
C	Century	Returns the number of completed days this century 1 (*)
D	Days	Returns the number of days so far this year in the format ddd
E	European	Date in the format: dd/mm/yy
J	Julian	Date in the format: yyddd (*)
M	Month	Returns the English name of the current month (**)
N	Normal	Date in the format: dd mon yyyy
O	Ordered	Date in the format: yy/mm/dd
S	Standard	Date in the format: yyyymmdd
U	USA	Date in the format: mm/dd/yy
W	Weekday	Returns the English name of the current day of the week (**)

The 11 available date formats are listed here with their descriptions. Any of these formats can be used for `date_format1` or `date_format2`.

* - When used for `date_format1`, this format is only valid when `input_date` is not specified.
** - This format is only valid for `date_format1`.



Option	Meaning	Example	
		DATE()	/* 1 May 2008
	Normal	DATE(,'01/27/08','Usa')	/* 27 Jan 2008
B	Base	DATE('B')	/* 730240
C	Century	DATE('C')	/* 122
D	Days	DATE('Days')	/* 122
E	European	DATE('Euro','122','B')	/* 01/05/08
J	Julian	DATE('JULIAN')	/* 00122
M	Month	DATE('Month')	/* May
N	Normal	DATE('N','01/27/08','U')	/* 27 Jan 2008
O	Ordered	DATE('O')	/* 00/05/08
S	Standard	DATE('S')	/* 20080501
U	USA	DATE('usa')	/* 05/01/08
W	Weekday	DATE('WEEKDAY')	/* Monday
		DATE('Wednesday')	/* Monday

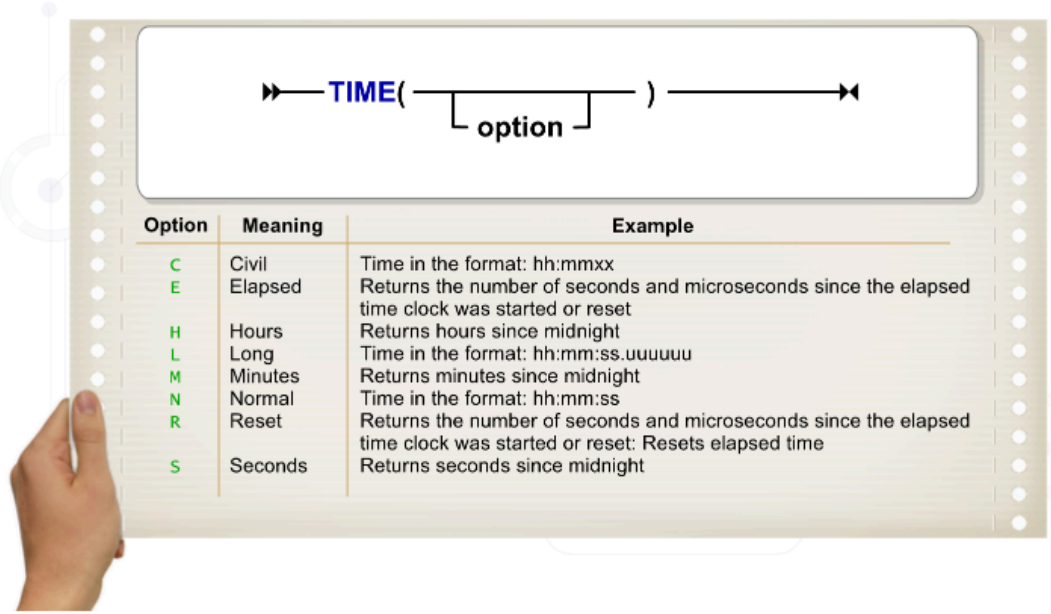
Shown above are examples of the DATE function options and their results. Where a date is not specified, assume a date of May 1, 2008. Dates can also be calculated by converting a date to the base date, adding or subtracting a number, and converting the date back, for example:

```
d1 = DATE("B")+30 /*convert today's date to a Base date and adds 30*/
d2 = date("N",d1,"B") /* convert back to a Normal from Base format */
```

This is the same as:

```
d2 = DATE("N",DATE("B")+30,"B")
```

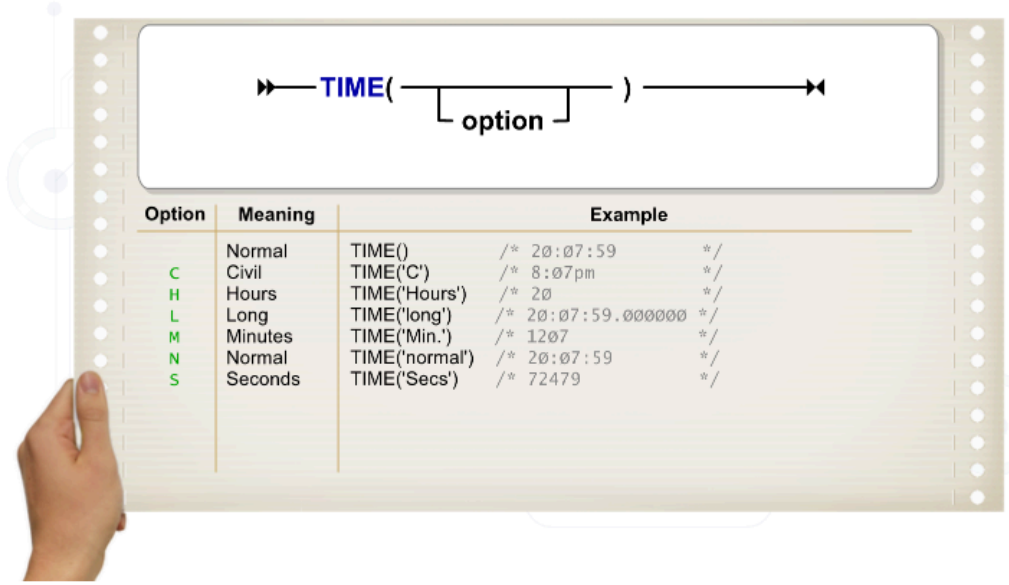
Note: `date_format` parameters ignore any characters after the first character; however, specifying the value shown in the "Meaning" column helps to document the program; `date_format` parameters are not case-sensitive, but they can be variables.



Option	Meaning	Example
C	Civil	Time in the format: hh:mmxx
E	Elapsed	Returns the number of seconds and microseconds since the elapsed time clock was started or reset
H	Hours	Returns hours since midnight
L	Long	Time in the format: hh:mm:ss.uuuuuu
M	Minutes	Returns minutes since midnight
N	Normal	Time in the format: hh:mm:ss
R	Reset	Returns the number of seconds and microseconds since the elapsed time clock was started or reset: Resets elapsed time
S	Seconds	Returns seconds since midnight

The TIME function without an option returns the time in 24-hour clock format (hh:mm:ss).

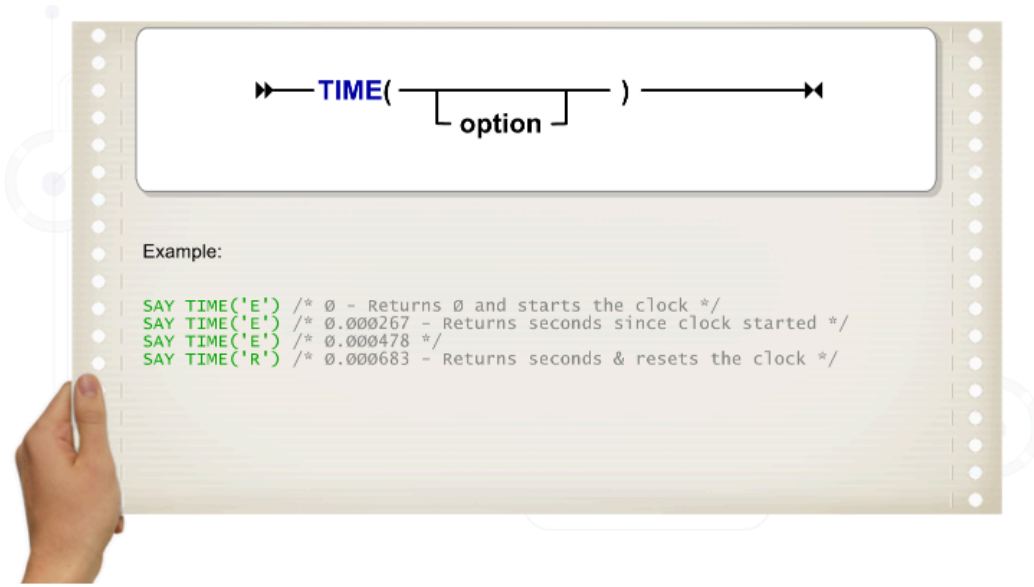
The Option parameter can be used to return the time in alternative formats, or to control and query the elapsed-time clock.



Option	Meaning	Example	
	Normal	TIME()	/* 20:07:59 */
C	Civil	TIME('C')	/* 8:07pm */
H	Hours	TIME('Hours')	/* 20 */
L	Long	TIME('long')	/* 20:07:59.000000 */
M	Minutes	TIME('Min.')	/* 1207 */
N	Normal	TIME('normal')	/* 20:07:59 */
S	Seconds	TIME('Secs')	/* 72479 */

These examples of the TIME function options assume a time of 20 hours, 7 minutes, 59 seconds.

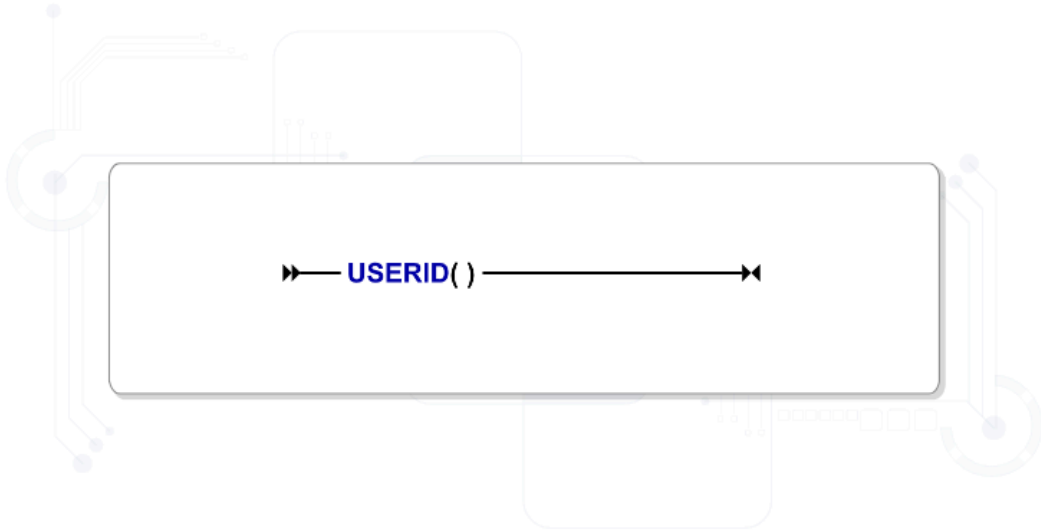
If used, the first character of Option is the only character that is required. Any characters that follow it are ignored, but specifying the full meaning helps to document your program. Option is not case-sensitive.



The TIME function can also be used to measure elapsed or real time intervals, which are accurate to one microsecond.

When TIME is first called by using the E or R option, it returns 0 and starts the real-time clock. Calling TIME again by using R returns the elapsed time since the E or R option was last used, and resets the real-time clock to 0. Subsequent calls to TIME by using E return the elapsed time since the first call, or since the last call by using R.

The above example shows a series of TIME functions. A real program would time something, such as how long it took to input an answer to a question.



The USERID function returns the logon ID of a user running in the TSO/E address space. In non-TSO/E environments, the result could depend on the requirements of the data center. The USERID function has no parameters.

```
USERID() /* USER1 - perhaps! */
```

Note: `USERID` is a non-SAA built-in function and is currently only available under TSO/E and VM.