z/OS Host Command Environments

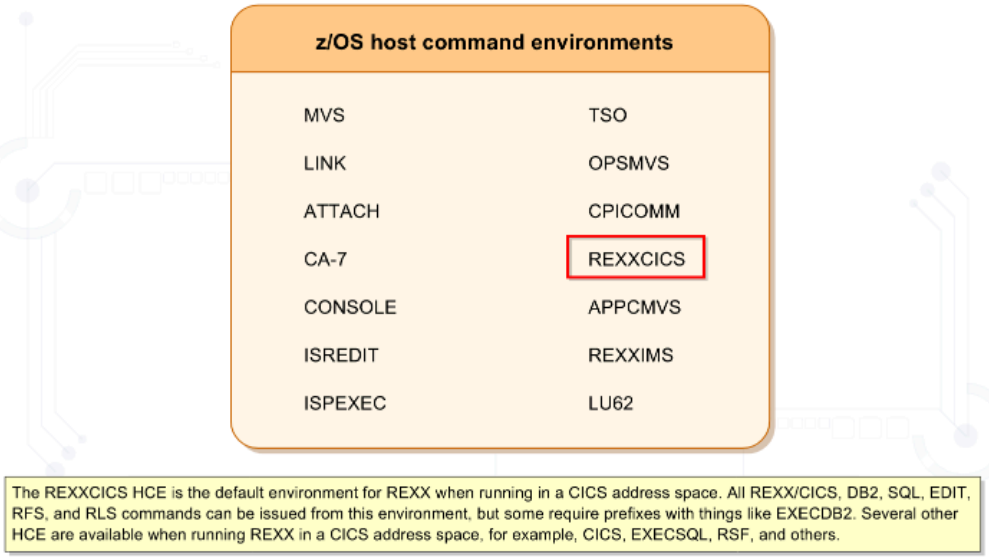
Objectives

z/OS Host Command Environments

In this module, you will examine some of the IBM and third-party host command environments available in the z/OS platform.

After completing this module, you will be able to:

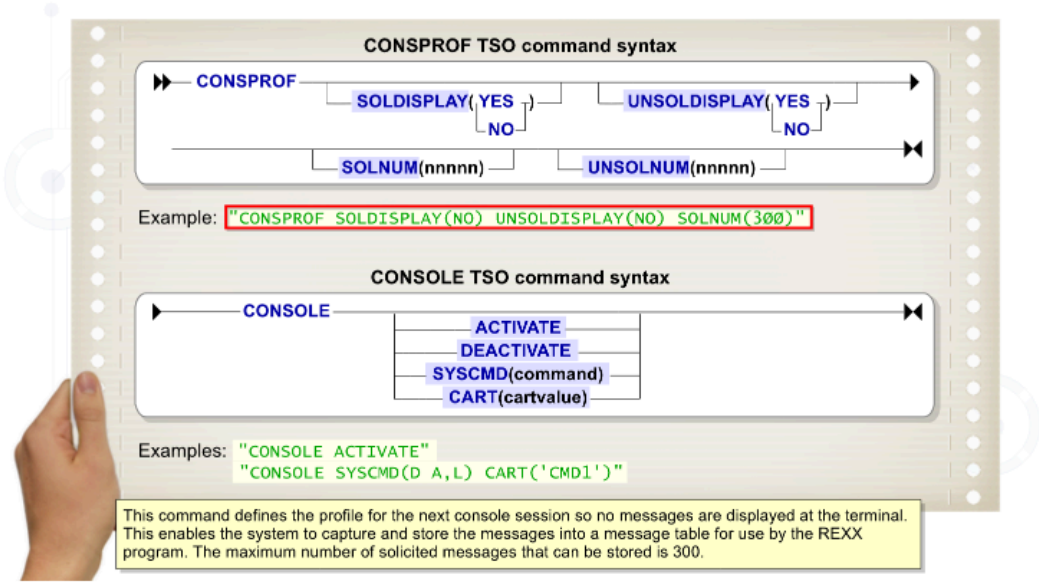
- Identify How to Start and Use the CONSOLE HCE
- Identify How to Execute ISPF Services By Using the ISPEXEC and ISREDIT HCE
- Identify How to Start and Interface with CA-7 By Using the CA-7 HCE



Every platform that runs the REXX interpreter has its own set of host command environments (HCE) that enable REXX programs to interface with base facilities and products running on the system. Although z/OS comes with several standard HCE, many third-party vendors also supply HCE so that REXX can interface with their products. These must be defined to the system by the systems programmer before they can be used. Some HCE also require the associated product to be running in the same address space.

Listed above are some of the HCE available on z/OS systems. You will explore the CONSOLE, ISPEXEC, ISREDIT, and CA-7 HCE.

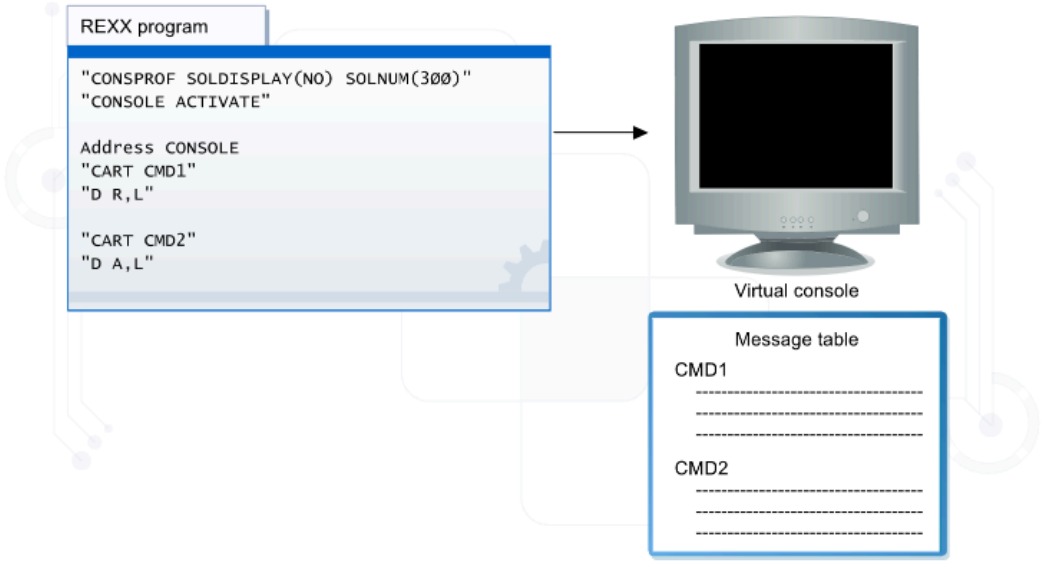
Mouse-over each HCE for a description.



The CONSOLE HCE enables z/OS system console commands to be executed from a REXX running in a TSO address space, and their output to be interrogated. Before executing the ADDRESS CONSOLE instruction, a profile should be defined for the session with the CONSPROF TSO command.

A console session must then be started by using the CONSOLE ACTIVATE TSO command before the ADDRESS CONSOLE instruction can be executed. If only one command is required to be executed, the CONSOLE command parameters can be used to execute the command and receive the response in a defined command and response token (CART).

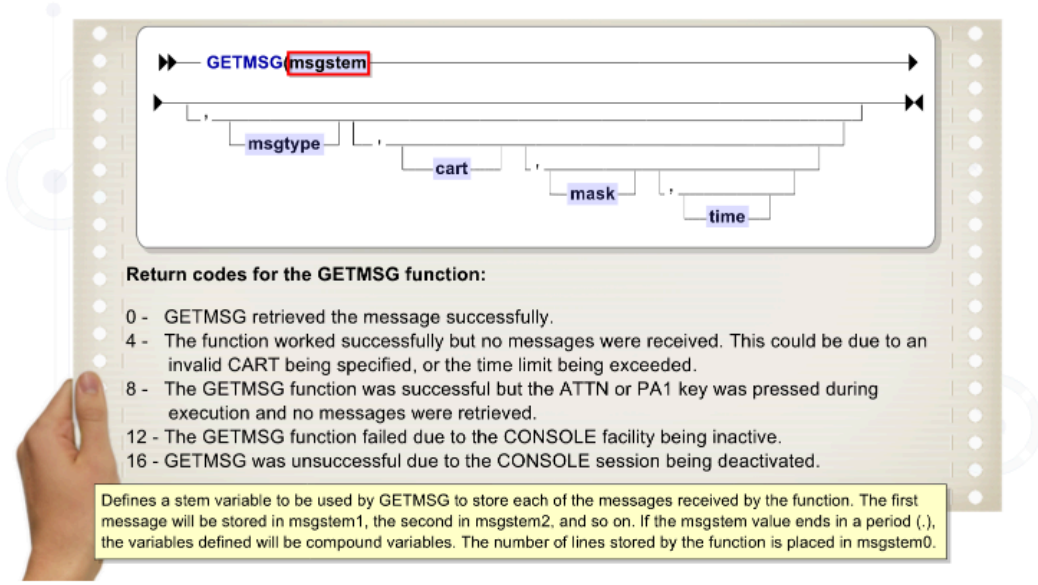
Mouse-over the CONSPROF and CONSOLE command parameters for descriptions of their use.



Although system commands can be executed from the TSO/E CONSOLE command, a REXX program can use the ADDRESS CONSOLE environment after it has been activated.

Two basic types of commands are entered from the ADDRESS CONSOLE instruction. The first is any system command, such as z/OS or JES, that would normally be entered on a system console. The second is a CART value command that enables responses from any subsequent commands to be identified by a CART of the name `value`, indicating which command the message is from. In this way, multiple commands can be entered and their responses individually identified; `value` can be 1-8 characters in length.

Click **Play** to see some examples of the CONSOLE environment.



Now that commands have been executed and their message responses saved in separate CARTs, the GETMSG function can be used to read the messages from the message table.

GETMSG is a special TSO/E external function that only returns a return code that indicates the success or not of the function. If the function is successful, it sets a number of compound variables, each containing one line of the messages received from the associated command. The CART can be used to distinguish which messages come from which command.

Mouse-over the syntax of the GETMSG function for descriptions of its parameters.

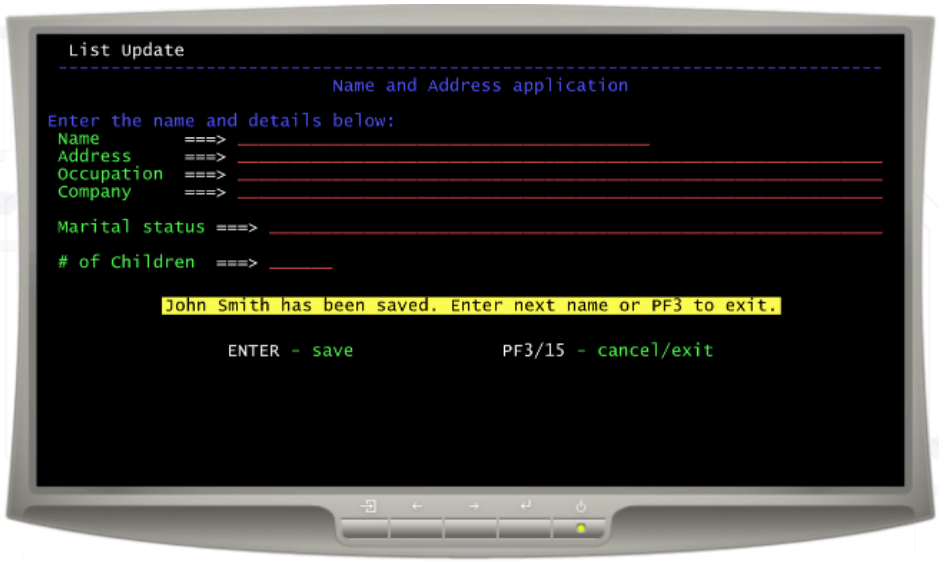
```
EDIT       USER1.REXX.EXEC(CONSEXEC)                          Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ********************************
000001 /* REXX - This program executes system console commands */
000002 Address TSO
000003 "CONSPROF SOLDISPLAY(NO) UNSOLDISPLAY(NO) SOLNUM(300)"
000004 "CONSOLE ACTIVATE"
000005 "CONSOLE SYSCMD(D A,L) CART('CMD1')"
000006 Address CONSOLE
000007 "CART CMD2"
000008 "D R,L"
000009 "CART CMD3"
000010 "D TS,L"
000011 G1 = GETMSG('ACT.','SOL','CMD1',,10)
000012 do a1 = 1 to act.0
000013    say act.a1
000014 end
000015 G2 = GETMSG('rep.','SOL','CMD2',,10)
000016 do r1 = 1 to rep.0
000017    say rep.r1
000018 end
000019 G3 = GETMSG('TSO.','SOL','CMD3',,10)
000020 Address TSO "CONSOLE DEACTIVATE"
```

Get all solicited response messages associated with the 'CMD2' CART and place them into the compound variables REP.n. Note that sol is not in quotes. As uninitialized REXX variables have a value of the variable name in uppercase, this will translate to the SOL and be accepted. However, this is not good practice. Again, G2 should be checked to ensure it has a value of 0.

This example shows how the CONSOLE environment can be used in a REXX program.

The CONSOLE command and system commands require specific authorization through the systems security package before they can be used. This must be set up by the systems programmer and security administrators.

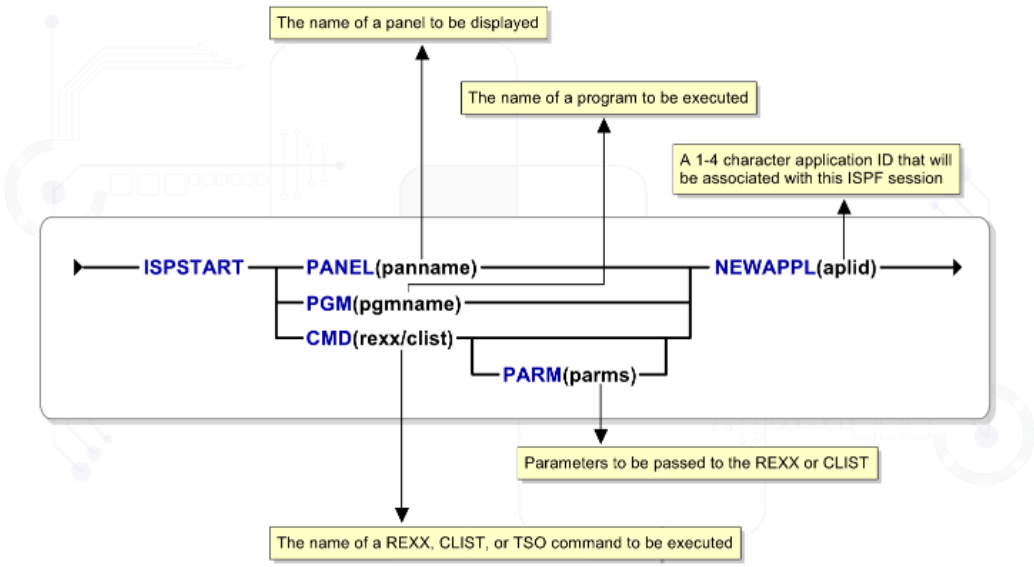
Mouse-over each line of code for a description.



A standard REXX uses unformatted screen text to read and write data from the terminal. Generally, every PULL statement is preceded by a SAY statement requesting and describing the information required. Applications that require large amounts of data to be entered can become cumbersome because only one set of information can be entered at a time.

ISPF has a set of services or commands that can be used by REXX to interface to several ISPF facilities, such as formatted screen panels, two-dimensional tables, and file-tailoring routines.

Click **Play** to see the difference between unformatted and formatted conversations with REXX.



The ISPF product must be started before any ISPF service can be used. In most installations, ISPF is automatically started when an online TSO session is activated, but it must be started with a command when running TSO in batch.

The ISPSTART command is used to initiate ISPF. There are many possible parameters but the main ones are PANEL(panname), PGM(pgmname), and CMD(rexx/clist). Each of these are mutually exclusive and they determine where a panel is to be displayed, a REXX or CLIST is to be executed, or a program is to be run.

Some installations have an ISPF command that is really a REXX or program that executes ISPSTART PANEL(ISR@PRIM) to display the ISPF Primary Option menu. A few of the ISPSTART parameters are shown above.

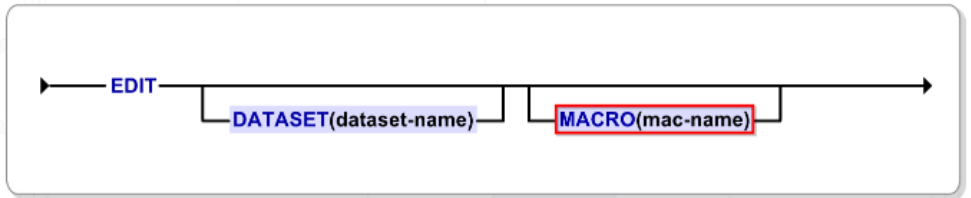
```
EDIT       USER1.REXX.EXEC(NAMESSPF)                          Columns 00001 00072
Command ===> ____________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ********************************
000010 /* REXX - This program Adds names to a table */
000020 Address ISPEXEC
000030 "TBOPEN NAMES"
000040 do forever
000050    "DISPLAY PANEL(NAMEADD)"
000060    if rc \= 0 then
000070       do
000080          "TBCLOSE NAMES"
000090          EXIT
000100       end
000110    "TBADD NAMES"
000115    "SETMSG MSG(NAM001)"
000120    name = ''
000130    addr = ''
000140    company = ''
000150    occup = ''
000160    marstat = ''
000170    kids = ''
000180 end
```

If Enter was pressed, add the data to the table. The variables and values that are added were determined when the table was created by the TBCREATE service.

After ISPF has been activated, ISPF services can be executed from a REXX by using the ADDRESS ISPEXEC keyword instruction. There are approximately 100 ISPF services so you will just look at how they are executed.

The above code describes how ISPF services are executed to display the previous example of a REXX using formatted screens.

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

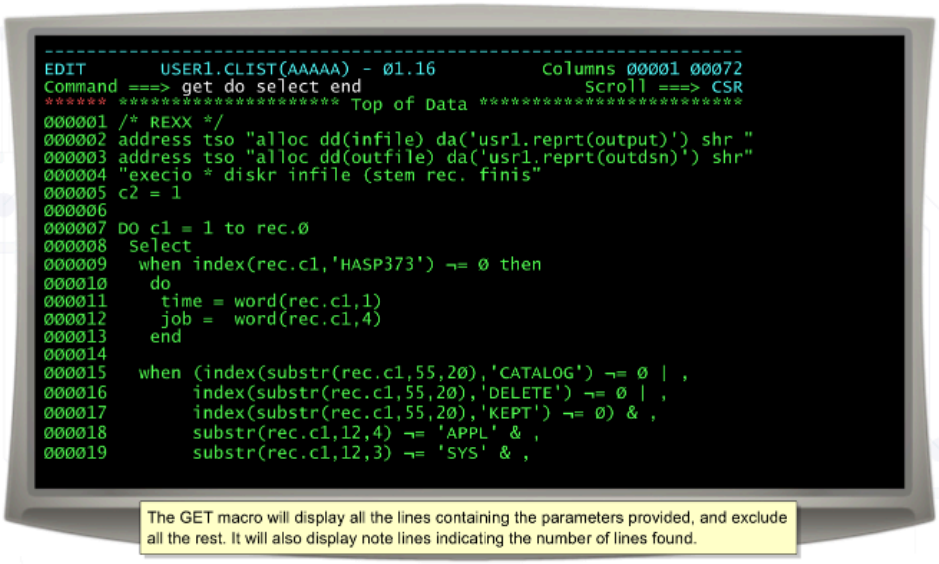


The name of an edit macro can be included when executing the EDIT service. This will cause the specified macro to be executed as soon as the edit facility is opened. It is possible for an edit macro to save the file and close it after executing whatever processing is required, and returning to the original REXX program.

Another major use of REXX is the creation of edit macros. While running the ISPF Edit facility, a REXX can be executed in the same way as a normal command, and perform virtually any task that can be done manually, including inserting, deleting, and changing lines.

To enter ISPF Edit, use the standard ISPF menu options or the ISPF EDIT service. The syntax of the common ISPF EDIT service parameters is shown above. Other parameters are available but are not covered here.

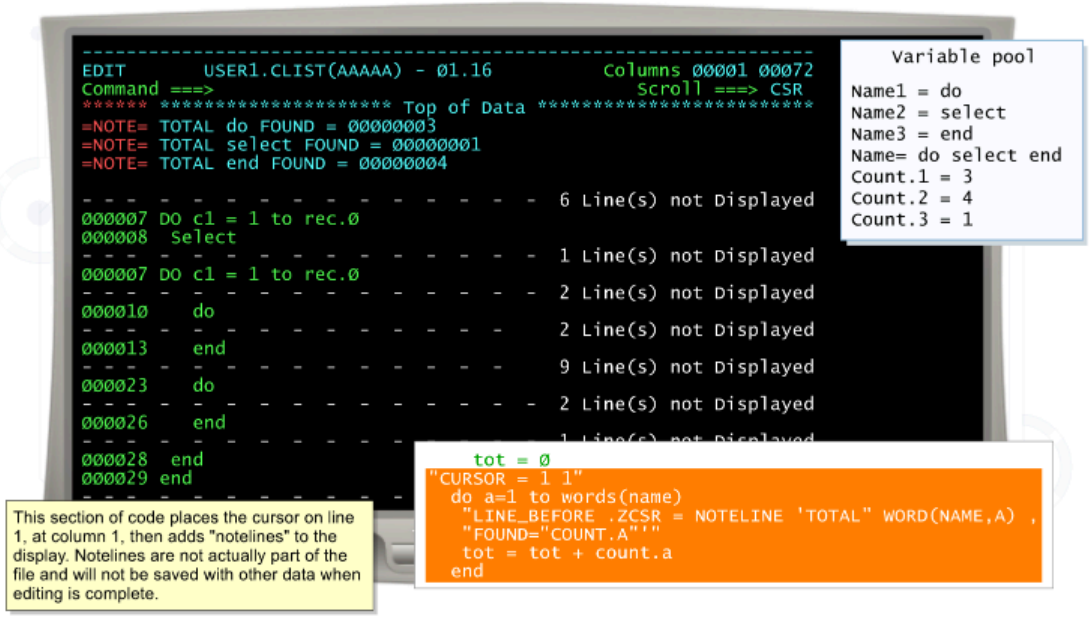
Mouse-over each parameter for a description.



After the EDIT facility has been entered, any REXX that is a member of one of the data sets allocated to SYSEXEC or SYSPROC can be executed by entering its name on the command line. However, the MACRO command must be entered in the ISREDIT host command environment (HCE) to execute the REXX as an EDIT macro, that is: `ADDRESS ISREDIT "MACRO"`

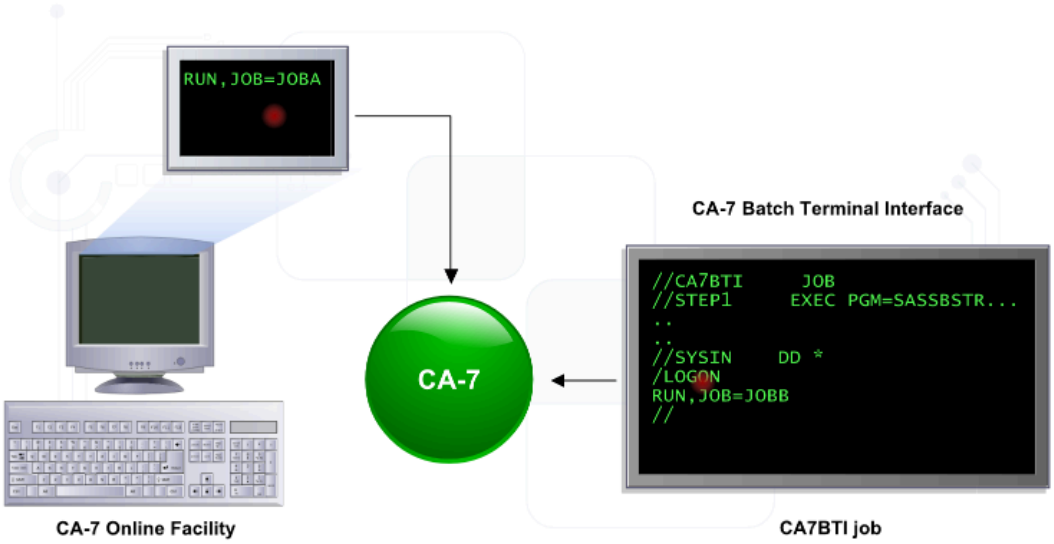
After MACRO has been specified, any one of approximately 100 ISREDIT commands can be entered to interrogate and manipulate the current data set. Note that parameters passed to an EDIT macro are not recognized by the ARG or PARSE ARG instruction, but must use the MACRO PARMS ISREDIT command instead. For more information, refer to the IBM Edit and Edit Macros manual.

Click **Play** to see how an EDIT macro can be executed from ISPF Edit.



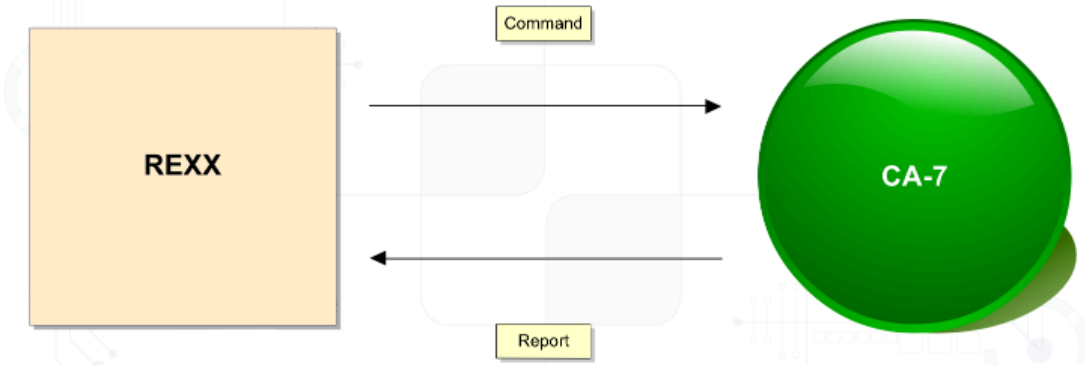
This code was used to produce the end result from the previous page. Each macro instruction could have been entered manually to produce the same result.

Click Play to see how each part of the macro would work if performed manually.



Vendor products on z/OS often have their own HCE environments. Computer Associates CA-7 scheduling product includes a REXX interface facility to enable users of the REXX programming language to communicate with and maintain CA-7 databases.

To communicate with CA-7, REXX uses the Common Communications Interface (CCI), which is described in the CA-7 Interfaces Guide.



The CCI interface enables a REXX user to issue commands to CA-7 and receive CA-7 output. All output returned by CA-7 is in the CA-7 batch format, that is, the same format as BTI output.

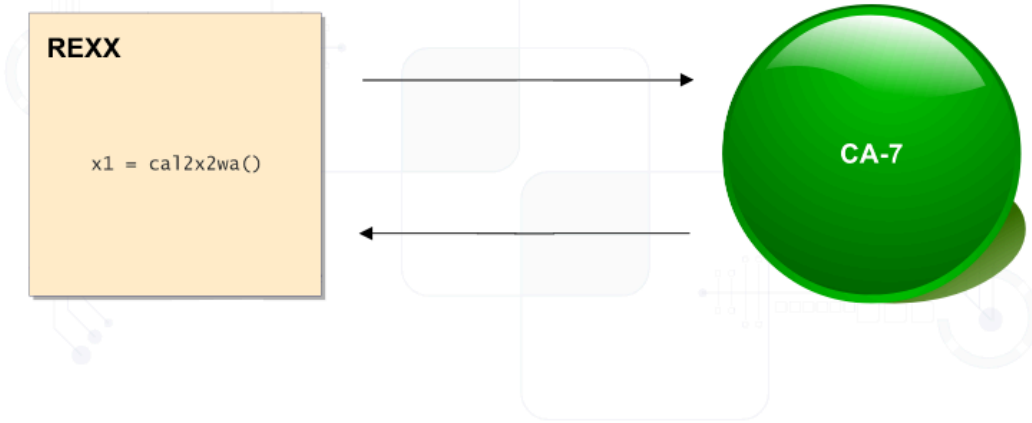
Click Play to see how REXX communicates with CA-7.

CCI REXX interface functions	
CAL2X2WA	Used to initialize the REXX address environment
CAL2X2WR	Used for the REXX command interface

Two CCI programs can now be used by REXX to interface with the CA-7 HCE.

The CAL2X2WA function is used to initialize the CA-7 address environment. The CAL2X2WR function is used for the REXX command interface.

You will focus on the CAL2X2WA function.

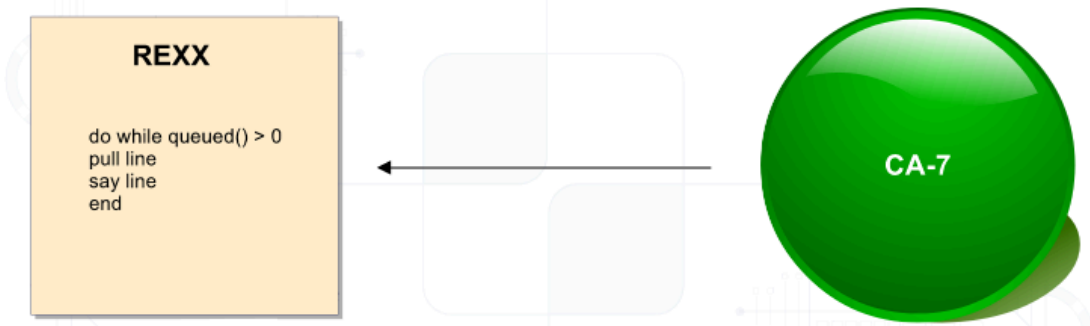


When REXX communicates with CA-7, the CA-7 HCE must first be initialized by using the CAL2X2WA function. This function has no parameters and is usually coded as an assignment statement, for example:

```
x1 = cal2x2wa()
```

Commands can now be sent to CA-7 by using the ADDRESS CA7 keyword instruction. CA-7 must be active for the environment to initiate successfully.

Click **Play** to see how REXX initializes the CA-7 address environment and sends commands to CA-7.



After entering CA-7 commands by using the ADDRESS CA7 instruction, any output produced by these commands is returned to the REXX external data queue or stack.

These lines can be read by the REXX program by using the PULL keyword instruction. If the command fails, the REXX system variable RC will contain a non-zero return code.

Click Play to see the code that REXX uses to send commands and receive output from CA-7.

```
/* DEMANDH REXX */
/*********************************************************************/
/* This REXX will check whether a specific job is in the request     */
/* queue and, if not, it will DEMAND it in with a hold status.       */
/* Execution syntax: DEMANDH jobname schedule-id                     */
/*********************************************************************/
arg testjob schid                                    /*Get Parameters*/
if schid = '' then schid = 1                        /*Set default SCHID*/
x1 = cal2x2wa()                  /* initialize CA7 address environment*/
found_flag = "no"                                  /* set job found flag*/
address CA7 "LREQ,JOB=*"                    /* sent LREQ command to CA-7*/
if rc <> 0 then                           /* test return code from command*/
 do
  say "LREQ command failed. Return Code = "rc
  exit                                          /* exit if command failed*/
 end
```

Shown above and on the next two pages is a sample REXX that can be executed from TSO to demand a job into CA-7 if it does not already exist in the request queue.

The first section of this REXX sets the job name and schedule ID from parameters entered when the REXX is executed. It then initializes that CA-7 address environment and sends an LREQ command to CA-7.

All commands should be tested for a non-zero return code to ensure that the command has worked successfully.

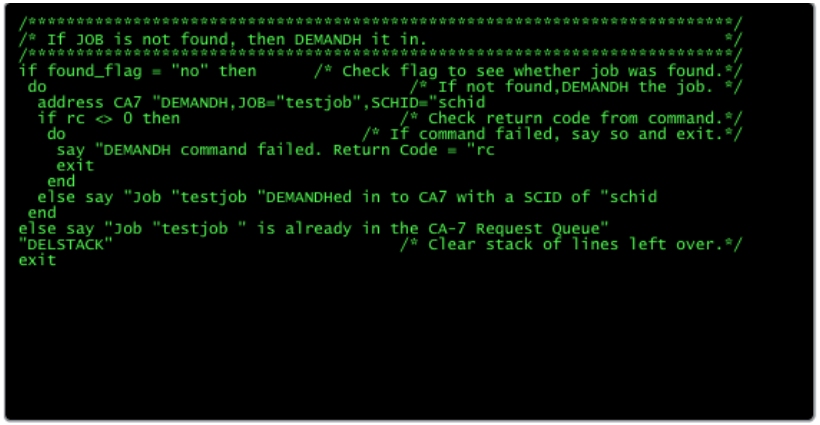
```
1LREQ
 JOB=TESTJOB                                              DATE=YY.DDD     PAGE 0001

   JOB  QUEUE  CA-7  -DAY(DDD) AND TIME(HHMM)--    CPU      SCH ENTRY MSTR  JOB
   NAME  NAME  NUM   DEADLINE SUB/START DUE-OUT SPEC/RUN  ID  MODE  REQ  STATUS

 TESTJOB  REQ  0016  123/1262 123/1362 123/1462  SY2-     001 DEMD  000  LATE

 SLIF-00 REQUEST COMPLETED AT 12:32:43 on YY.DDD
```

```
/*********************************************************************/
/* Read output and test for required job.                            */
/*********************************************************************/
 do while queued() > 0                          /*Set loop to read all lines.*/
  pull listline                                          /* Get next line. */
  if strip(substr(listline,2,8)) = testjob then    /*Test line for jobname.*/
   found_flag = "yes"                                 /*If found,set flag.*/
 end
```

The output received from a successful LREQ command would look like the output shown at the top of the screen. This is returned in the same format as would be produced by the CA-7 batch terminal facility, and it has carriage control characters in column 1.

This section of the REXX reads every line of the output and looks for the requested job name in columns 2-10, stripping off any trailing blanks.

If the job name is found, the job found flag is set to "yes".



```
/*********************************************************************/
/* If JOB is not found, then DEMANDH it in.                          */
/*********************************************************************/
if found_flag = "no" then      /* Check flag to see whether job was found.*/
 do                                        /* If not found,DEMANDH the job. */
  address CA7 "DEMANDH,JOB="testjob",SCHID="schid
  if rc <> 0 then                    /* Check return code from command.*/
   do                              /* If command failed, say so and exit.*/
    say "DEMANDH command failed. Return Code = "rc
    exit
   end
  else say "Job "testjob "DEMANDHed in to CA7 with a SCID of "schid
 end
else say "Job "testjob " is already in the CA-7 Request Queue"
"DELSTACK"                                 /* Clear stack of lines left over.*/
exit
```

This section of the REXX checks to see whether the requested job was found in any of the output. If not, the job will be demanded into CA-7 by using the DEMANDH command and a message displayed. If the job is found, a message will also be displayed.

It is good practice to test return codes from commands and clear the REXX external data queue or stack before exiting a REXX that uses the CA-7 HCE.