

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

30-6-2021

# Entorno HOST

Guía de uso del entorno HOST,  
COBOL y otras cosas relacionadas.

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

MARÍA INMACULADA CAMPILLO SOTO

## ÍNDICE

Control de versiones .....	1
Objetivo de la documentación .....	1
COBOL .....	2
Introducción a COBOL .....	2
Divisiones .....	2
IDENTIFICATION DIVISION.....	2
ENVIRONMENT DIVISION.....	2
DATA DIVISION .....	2
PROCEDURE DIVISION.....	3
Variables.....	3
Campos de trabajo.....	4
Literales .....	4
Constantes .....	5
Switches .....	5
Operaciones .....	5
Suma.....	5
Resta.....	5
Multiplicar .....	6
Dividir .....	6
Mover (asignar) .....	6
Condicionales .....	6
IF/END-IF .....	6
EVALUATE.....	6
Rutinas y ciclos .....	7
Variables especiales .....	7
Índices .....	7
Arreglos/tablas de datos .....	7
Tablas .....	8
Consultas con DB2.....	8
Cursores .....	8
Consultas a otros servicios.....	9
HOST.....	10
Atajos de teclado/Comandos.....	10
PARAR UN SERVICIO.....	11
Conceptos avanzados.....	12

Trabajos.....	12
Información sobre una partición .....	12
Sintaxis de JCL .....	13
Inicio.....	14
Paso .....	14
Ejecución del JOB .....	16
Procesos .....	16

## Control de versiones

Número de versión	Fecha
1 – creación	2021/06/30

## Objetivo de la documentación

En esta guía encontrarás, principalmente, lo necesario para obtener los conocimientos básicos del lenguaje de programación COBOL y de su entorno, conocido como HOST.

La primera contiene información sobre el propio lenguaje, incluido la definición de variables y la estructura del código. En la segunda parte se encuentra tanto los atajos de teclado como la información básica de uso del entorno HOST.

Además, los siguientes apartados recogen información más concreta servicios y todo lo referente a ellos.

## COBOL

### Introducción a COBOL

COBOL es un lenguaje de programación orientado a un lenguaje de negocio (COmmon Business-Oriented Language). Sus características, entre otras, tenemos que COBOL procesa grandes cantidades de información y maneja de manera efectiva las excepciones y los errores. Además, es un lenguaje de programación de alto nivel y estructurado con operaciones en cascada. Esto último se refiere a que se ejecuta de manera que el inicio de cada etapa debe esperar a que la anterior finalice.

Es de aclarar que los programas que se escriben en COBOL se dividen en dos secciones: componentes y servicios. Sin embargo, la distinción la haremos en los posteriores apartados, aunque tienen la misma estructura. Para comenzar con los conocimientos básicos de COBOL los llamaremos programas o software.

### Divisiones

Cualquier programa escrito en COBOL cuenta con 4 divisiones principales: la división de identificación, la del entorno, datos y procesos. En cada uno de ellos se guarda diferentes tipos de información.

A continuación, describiremos cada uno de ellos. Lo pondremos con el nombre tal cual aparecen en los programas/software, es decir, en inglés. Es importante destacar y como veremos que los . (puntos) son importantes en COBOL. Por tanto, cada división y sección debe terminar con un punto.

#### IDENTIFICATION DIVISION

En esta primera parte se encuentra los datos de identificación: quién ha hecho el programa y cómo se llama, entre otras cosas.

#### ENVIRONMENT DIVISION

Aquí es donde se pone la información de compilación y si se necesita archivos, es en esta sección donde se declaran. Además, se pone otros tipos de configuración.

En la práctica esta sección siempre se deja vacía.

#### DATA DIVISION

En esta división es donde declaramos las variables, constantes, tablas, etc. Asimismo, se divide en dos subsecciones: WORKING-STORAGE SECTION (es la sección concreta donde declaramos todo lo dicho anteriormente) y LINKAGE SECTION (aquí se declara la *copy*<sup>1</sup> propia del programa)

---

<sup>1</sup> Este concepto se verá en otro apartado

## PROCEDURE DIVISION

Esta última división es la que tiene el código dividido en párrafos. Generalmente<sup>2</sup>, tiene los siguientes párrafos iniciales: de inicio (inicialización de variables), proceso (la lógica principal del programa) y de fin (manda una respuesta de si ha ido bien o no el programa).

Además, esos párrafos pueden llamar a otros, esto es como en otros lenguajes de programación, llamadas a diferentes funciones.

### Variables

Como hemos dicho antes, es en la parte de WORKING-STORAGE SECTION donde debemos declarar las variables.

Hay diferentes tipos de variables: las que usamos como campos de trabajo (se almacenan valores de los pedidos), índices<sup>3</sup>, variables de error, literales, constantes y switches.

Estos se organizan en diferentes niveles, el '01' se usa para los tipos dichos anteriormente. Para el resto tenemos que '02' al '49' son otros niveles que pueden ser divididos en otros niveles, es decir como variables divisibles. Sin embargo, usaremos de '05', '10'.... El nivel '77' se usa para variables no divisibles, pero en la práctica está en desuso. Por otra parte, está el nivel '88' que se reserva para los switches.

Además, todas las variables se deben declarar y, en manera de lo posible, inicializar en su correspondiente apartado. COBOL maneja tres tipos de dato: numerales, de tipo carácter y varchar (pueden ser números y caracteres).

Para declarar una variable usaremos la siguiente notación:

05 WS-VARIABLE      PIC X(12) VALUE SPACES.

05: es el nivel

WS-VARIABLE: es el nombre de la variable, la cual depende del tipo de variable empezará por WS, LT, entre otras cosas. Esto los veremos en la definición de cada tipo de variables.

PIC: está es como el símbolo de asignación en otros lenguajes de programación. Sin embargo, solamente se usa para las declaraciones en la sección de WORKING-STORAGE SECTION.

X: aquí es donde se hace referencia al tipo de dato, si es un número, un carácter o ambas cosas. Usaremos X para los que sean de ambos tipos, A para los que son caracteres (puede ser uno solo o una cadena) y 9 para los números. COBOL es capaz de diferenciar si un número es positivo o negativo si le ponemos S9 en vez de 9. Además, si queremos que tengan decimales podemos poner 9(V99 para dos decimales o 9(V(2), por ejemplo. Aunque tanto S9 como los decimales están en desuso.

12: es la longitud de la variable. Para COBOL es necesario poner qué longitud tendrá la variable, la inicialicemos o no.

---

<sup>2</sup> Los servicios se dividen en servicios orquestables y no orquestables donde los primeros suelen tener en los párrafos iniciales otro más, pero estos los veremos en otro apartado.

<sup>3</sup> Los índices los veremos después del apartado de Arreglos/tablas de datos, puesto que se usan para estos tipos de datos.

**VALUE SPACES:** esta parte puede variar según cada caso. Aquí se inicializa, puede ser a espacios (**SPACES**), un valor concreto (**'CADENA'**, **1** → como un número) o sin inicializar. Cuando lo hagamos debemos poner **VALUE** y luego el valor correspondiente.

Tanto se inicialice como no, debemos poner un punto (.) al final de la línea, tal y como aparece en el ejemplo. Esta parte es MUY importante, puesto que COBOL diferencia a cada una de las secciones diferentes mediante este símbolo.

A continuación, describiremos detalladamente cada uno de los tipos, su uso y algunos ejemplos.

### Campos de trabajo

Un servicio, uno de los tipos de programas, trabaja con datos que vienen ya dados y debemos guardarlos en ese sitio. Como referencia a otros lenguajes de programación, son los argumentos que se pasan al ejecutar un programa. Sin embargo, en COBOL los debemos coger de la *copy* propia del servicio y guardarlos.

Todos estos valores los guardaremos en este tipo de variables. Por ejemplo:

#### 01 CAMPOS-TRABAJO.

05 WS-VARIABLE1                      PIC X(12) VALUE SPACES.

05 WS-VARIABLE2                      PIC X(6) VALUE SPACES.

05 WS-VARIABLE3                      PIC S9(2) COMP.

05 LT-M                                PIC X(1) VALUE 'M'.

Como puedes observar, el primer nivel es el '01' que se llama CAMPOS-TRABAJOS y tiene un punto (.) al final. El resto son subniveles, '05', las diferentes variables con sus nombres empezado por WS (WORK SECTION). Cada una de ellas está declarada con diferentes tipos de datos y, algunas de ellas, inicializadas a espacios. Sin embargo, todas terminan en punto.

Como podemos observar, algunas empiezan por WS, puesto que van a ser variables que cambian de valor, y otras por LT, esto es porque son literales, es decir, cadenas de caracteres que no cambiarán su valor.

Por otra parte, tenemos la palabra reservada COMP hace referencia a *comprimido*, a nivel memoria ocupa menos puesto que la respuesta es un número largo esto ocupa bastante.

### Literales

Los literales son variables cuyo valor es un carácter o una cadena, además, como principal característica tiene que ese valor no va a cambiar a lo largo de la ejecución del programa. Deberán empezar con LT para mostrar que la variable que estamos usando es un literal y no otro tipo.

Como apunte, otras veces encontrarás que este tipo también tiene declarados números, que, en vez de ser contantes, serán literales. No tiene diferencia ninguna en este caso, es propia de la implementación de cada programado. Sin embargo, se recomienda que los literales sean para los caracteres y las constantes para los números<sup>4</sup>.

---

<sup>4</sup> Sin embargo, para referirnos a los números de párrafos, usaremos los literales.

## Constantes

En el anterior apartado hemos adelantado que son las constantes. Estas son variables numéricas que no cambiarán su valor a lo largo de todo el programa.

El valor con el que se inicializa las constantes suelen ser números, sin ponerlos entre comillas simples, puesto que de lo contrario se tomarán como una cadena y dará error de compilación.

## Switches

Los switches se dividen en tres partes: el nombre general etiquetado con el nivel '01' y dos subniveles del tipo '88', como hemos dicho anteriormente. Como, por ejemplo:

```
01 SW-ERROR                PIC X(1) VALUE 'N' .
   88 SI-ERROR              VALUE 'S' .
   88 NO-ERROR              VALUE 'N' .
```

Se puede apreciar que el switch se llama *ERROR*. Sin embargo, al declarar el nombre le ponemos SW al principio junto con la inicialización correspondiente. Generalmente, será a 'N', es decir a 'NO', aunque puede haber casos en los que ese valor sea a 'S'. A continuación, tenemos el mismo nombre, pero con un SI/NO y su correspondiente valor S/N. Éstos representan, en sí mismos, el switch. Actúan como un interruptor al cual uno solo puede tener un valor TRUE y, por contra, el otro tendrá valor FALSE.

## Operaciones

Hay dos maneras de hacer operaciones, asignar el nuevo valor a una nueva variable o a una ya existente en la operación. Por ejemplo:

Variable3 = Variable1 + Variable2

Variable1 = Variable1 + Variable2

La última es la más común.

## Suma

La estructura de la suma es la siguiente:

**ADD** Variable1 **TO** Variable2 → que equivale a Variable2 = Variable1 + Variable2

**ADD** Variable1 **TO** Variable2 **GIVING** Variable3

→ Que equivale a Variable3 = Variable1 + Variable2

## Resta

La estructura de la resta es la siguiente:

**SUBTRACT** Variable1 **FROM** Variable2 → que equivale a Variable2 = Variable1 - Variable2

**SUBTRACT** Variable1 **FROM** Variable2 **GIVING** Variable3

→ Que equivale a Variable3 = Variable1 - Variable2



## Multiplicar

La estructura de la multiplicación es la siguiente:

**MULTIPLY** Variable1 **TO** Variable2 → que equivale a  $\text{Variable2} = \text{Variable1} * \text{Variable2}$

**MULTIPLY** Variable1 **TO** Variable2 **GIVING** Variable3

→ Que equivale a  $\text{Variable3} = \text{Variable1} * \text{Variable2}$

## Dividir

La estructura de la división es la siguiente:

**DIVIDE** Variable1 **TO** Variable2 → que equivale a  $\text{Variable2} = \text{Variable1} / \text{Variable2}$

**DIVIDE** Variable1 **TO** Variable2 **GIVING** Variable3 **REMAINDER** WS-RESTO

→ Que equivale a  $\text{Variable3} = \text{Variable1} / \text{Variable2}$

## Mover (asignar)

**MOVE** Variable1 **TO** Variable2 → que equivale a  $\text{Variable2} = \text{Variable1}$

## Condicionales

### IF/END-IF

Para este tipo de instrucción tenemos el IF, aunque cada sentencia IF deberá ir con END-IF. Además, para comparar tenemos los siguientes símbolos y su equivalente en texto. Sin embargo, para hacer las comparaciones de igualdad es recomendable la forma en texto y dejar los símbolos para las sentencias SQL.

=	EQUAL
NOT =	NOT EQUAL
>	GREATER THAN
<	LESS THAN

Si queremos unir varias condiciones podemos usar las palabras AND y OR. Un ejemplo puede ser el siguiente:

```
IF WS-VALOR EQUAL 0 OR WS-VARIABLE2 GREATER THAN 4
  SET SI-ERROR TO TRUE
ELSE
  PERFORM PARRF
END-IF
```

### EVALUATE

Si diferentes valores para que una variable pueda tener, para la comparación de igualdad, podemos usar el bloque de código EVALUATE (con su correspondiente END-EVALUATE). Para cada posibilidad usamos la palabra reservada WHEN y para otro valor podemos usar WHEN OTHER.

```

EVALUATE WS-CATEGORY
  WHEN LT-ONES
    MOVE 1 TO IND
    MOVE 1 TO WS-VALUE
    PERFORM SUM-VALUE UNTIL IND > 5
  WHEN LT-TWOS
    MOVE 1 TO IND
    MOVE 2 TO WS-VALUE
    PERFORM SUM-VALUE UNTIL IND > 5
  WHEN LT-FULL
    CONTINUE
  WHEN LT-FOURK
    CONTINUE
  WHEN OTHER
    MOVE 0 TO WS-RESULT
END-EVALUATE

```

Además, podemos usar la palabra reservada CONTINUE en el caso de no querer hacer nada en una parte de IF o del CASE.

### Rutinas y ciclos

Para crear un bucle debemos crear un párrafo nuevo y llamarlo hasta que se cumpla cierta/s condición/es. Su estructura es la siguiente:

```
PERFORM SAME-VALUE UNTIL IND > 5 OR SI-FIN
```

En este caso, se llama al párrafo SAME-VALUE hasta que un switch sea cierto o el índice sea mayor que un número de ocurrencias dadas.

### Variables especiales

Hay algunas variables que hay que inicializarlas con la instrucción INITIALIZE, éstas son tablas y copys. Se suelen poner en un párrafo que se llama INICIO y en algunos servicios tiene el número 1000.

### Índices

Son variables como cualquier otra, situada normalmente en la sección de *indices*. Este tipo de variables son de tipo numeral inicializadas a 1. Su uso es, principalmente, para indicar en qué registro de tabla accedemos (número de ocurrencia).

```

01 INDICES.
05 WS-IND                                     PIC 9(1)  VALUE 1.

```

### Arreglos/tablas de datos

En COBOL, para tener un array/arreglo en un programa, debemos crear una tabla de manera “local” dentro de dicho programa. La declaración es parecida a la de una variable en un nivel superior (nombre de la tabla) y otras en el siguiente nivel (los campos de la tabla):

```

01 EMPLEADO OCCURS 10 TIMES.
02 NOMBRE PIC X(9) .
02 APELLIDO PIC X(9) .

```

Además, debemos saber cuántas ocurrencias (registros) va a tener con antelación. En el caso de acceder a un campo de esta tabla, debemos indicar el campo y el nombre de dicha tabla, por ejemplo: **NOMBRE OF EMPLEADO**

Al haber diferentes registros, para acceder a cada uno de ellos se usará un índice que será la posición a la que deseamos acceder entre paréntesis: **NOMBRE OF EMPLEADO(7)**

Esto último también se puede aplicar a las copys.

## Tablas

Las tablas se declaran en WORKING-STORAGE SECTION, pero se inicializan dentro del PROCEDURE DIVISION en el párrafo de inicio. De cualquier manera, siempre se declaran de la siguiente manera:

```

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE TABLA1 END-EXEC.
EXEC SQL INCLUDE TABLA2 END-EXEC.

```

La primera línea es obligatoria, aunque después declaramos por cada línea, una tabla diferente. Su estructura es simple EXCEL SQL INCLUYE [nombre de la tabla] END-EXEC.

## Consultas con DB2

En este proyecto, el lenguaje de la base de datos es DB2, bastante parecido a SQL.

## Cursores

Usaremos los cursores para aquellas consultas que devuelvan más de un registro y debemos tratar cada uno de ellos por separado. Normalmente acompañado de un switch para saber cuándo cerrarlo.

Los cursores se declaran en la sección de WORKING-STORAGE SECTION. Son en realidad consultas SQL que para usarlos necesitamos tener 3 párrafos diferentes: para abrir el cursor, leerlo y cerrarlo.

Tienen la siguiente estructura:

```

EXEC SQL
  DECLARE CURSOR-NOMBRE CURSOR FOR
  SELECT CAMPO_1
  FROM TABLA1
  WHERE CAMPO2 = :TABLA1.CAMPO2
END-EXEC.

```

Primero tenemos la instrucción de ejecución SQL: EXEC SQL – END-EXEC. Como hemos visto antes, al igual que el IF, debemos poner que se ha terminado esa sección. Además, como estamos en la parte de declaraciones, es obvio que es aquí donde se declaran con la palabra reservada DECLARE y a continuación, el nombre del cursor, generalmente, podremos:

CURSOR-[nombre]

Por último, será una *select* DB2 con los campos a consultar y filtrados por los datos necesarios.

En cuanto a los párrafos correspondientes para usarlos tenemos los siguientes:

#### *Abrir cursor*

Este párrafo se encarga de poner los valores correspondientes a las variables auxiliares usadas en la *select*, en otras palabras, podremos en el campo de la tabla a consultar el valor que correspondientes (el mismo campo que en el WHERE) y luego pondremos algo parecido a esto:

```
EXEC SQL
  OPEN  CURSOR-NOMBRE
END-EXEC.
```

Con la palabra reservada OPEN, abriremos el cursor para su uso.

#### *Usar cursor*

Este párrafo se llama de manera iterativa hasta que no hay más registros que procesar o encontremos uno válido (algunas veces esto se tiene en cuenta). Además, es recomendable el uso de un índice para saber por qué registro vamos. Sin embargo, aquí nos centramos en la ejecución SQL, que tiene esta estructura:

```
EXEC SQL
  FETCH CURSOR-NOMBRE
  INTO  :TABLA1.CAMPO_1
END-EXEC.
```

Con esto hacemos que se nos guarden en las correspondientes variables auxiliares (campos de la tabla) dentro de la tabla correspondiente. Después de controlar si hay algún error, tratamos los valores y si nos quedamos sin registros debemos activar su switch para luego cerrar el cursor.

#### *Cerrar cursor*

Al igual que en el párrafo de abrir, con la misma nomenclatura, tenemos una palabra reservada para cerrarlo: CLOSE.

```
EXEC SQL
  CLOSE CURSOR-NOMBRE
END-EXEC
```

### Consultas a otros servicios

Primero debemos poner la copy correspondiente del servicio al principio del WORKING-STORAGE SECTION:

```
01 COPY-NOMBRE.
COPY NOMBRE.
```

Van de dos en dos, la primera línea es el nombre con el que vamos a llamar al servicio y la segunda hace referencia a la propia copy de dicho servicio.

## HOST

### Atajos de teclado/Comandos

- Comandos:
  - **V**: de la palabra VIEW, es la manera de acceder a las carpetas y si queremos ver un programa en modo VIEW (las modificaciones que se hagan se borrarán).
  - **E**: de la palabra EDIT, se accede a un servicio/componente en el que queramos editar, los cambios se guardarán si salimos con **F3**.
  - **C**: si ponemos la C junto a un programa, podemos copiarlo a otra carpeta. Aparecerá una pantalla para poner la ruta nueva y ésta deberá ponerse entre comillas simples.
  - **M**: si ponemos la M junto a un programa, podemos moverlo a otra carpeta. Aparecerá una pantalla para poner la ruta nueva y ésta deberá ponerse entre comillas simples.
  - **D**: si ponemos la D junto a un programa, podemos eliminarlo de la carpeta actual.
  - **Z**: si ponemos Z sobre una carpeta, se comprime el contenido haciendo que el espacio sea menor.
  - **F palabra**: de la palabra FOUND, encuentra la primera coincidencia con dicha palabra. Útil en los programas. Con F5 irás navegando sobre la parte del programa donde está la palabra a buscar.
  - **L palabra**: de la palabra LIST, te lista la palabra que deseas buscar. Al contrario que FOUND, LIST se usa cuando lo que buscas está en una lista como la lista de servicios de una librería.
  - **M**: de la palabra MOVE y combinado con las teclas **F7/F8** te lleva al principio o final de la pantalla/programa.
  - **HI COBOL/JCL**: en los servicios/componentes en los que aparece todo en verde, debemos poner este comando en la línea de *command* para que aparezca el formato adecuado.
  - **START**: crea una nueva sesión en la máquina en la que estemos.
  - **SAVE**: cuando estemos editando un programa (servicio/componente) y deseamos guardar los cambios sin salir, escribimos este comando sin necesidad de salir para que se guarde (**F3**).
  - **PRE T\_\_\_\_\_\***: para cerrar la sesión de otro compañero cuando se queda "pillado", con este comando aparecerá su pantalla con todos los *logs*, entre ellos el de su sesión en blanco (significa que está en ejecución). Para cerrarla, pon un **P** delante y ENTER dos veces. Solo válido en integración.
- Teclado:
  - **F1**: muestra el menú de máquinas (MD, MP, MB...)
  - **F2**: ejecuta la query en la pantalla de TSO QMF(C), crea una sesión en paralelo sobre la misma máquina (o usando el comando **START**).
  - **F3**: sirve para salir de la pantalla actual (paso a paso) y guardar los cambios.
  - **F5**: cuando estás buscando algo con el comando *F*, te enseña uno por uno los elementos encontrados.
  - **F6**: en TSO QMF(C), ir para atrás/delante de la página de queries.
  - **F7/F8**: ambos para moverse arriba/abajo respectivamente de la pantalla. Si ponemos una M en la línea de comando te lleva al principio/final del programa.

- **F9**: sirve para moverse entre sesiones de la misma máquina y con **Re/Av pag** para diferentes.
  - **F10/F11**: moverse a la izquierda/derecha respectivamente en la pantalla.
  - **F12**: sirve para salir de la pantalla actual (paso a paso), pero sin guardar los cambios.
  - **Re/Av pag**: útiles para moverse entre sesiones de diferentes máquinas, al contrario que **F9** que es para la misma.
- En los servicios y componentes se puede dar los casos que quieras copiar, mover, eliminar... partes del código, para ello tenemos los siguientes comandos que deben ponerse sobre el número de línea (derecha) del servicio/programa:
- **D**: elimina dicha línea.
  - **DD**: a diferencia del anterior, este elimina un bloque de código. Debemos poner las dos D's al inicio y final del bloque.
  - **C**: copia la línea.
  - **CC**: copia una sección de código. Se ponen al principio y final del bloque. En el caso que copiemos de otro servicio/componente que no sea con el que estemos, se pone el comando **CUT** en la línea de COMMAND.
  - **M**: mueve la línea.
  - **MM**: al igual que los otros, mueve el bloque de código entre las marcas (al principio y final del bloque)
  - **A**: de la palabra AFTER. Se usa si hemos de pegar el código copiado o movido antes. Se situará en la línea de **después**. En el caso de que hayamos copiado código de otro sitio y deseemos ponerlo en nuestro servicio/componente, debemos poner **PASTE** en la línea de comandos.
  - **B**: de la palabra BEFORE. Se usa si hemos de pegar el código copiado o movido antes. Se situará en la línea de **antes**. En el caso de que hayamos copiado código de otro sitio y deseemos ponerlo en nuestro servicio/componente, debemos poner **PASTE** en la línea de comandos.
  - **I**: de la palabra INSERT, se añade una línea nueva, para seguir añadiendo ésta debe tener al menos un espacio en blanco si no tiene nada más.

## PARAR UN SERVICIO

En las pruebas unitarias, solemos pararnos el servicio en modo DEBUG.

### *Comandos en la pantalla del servicio*

- **SET AUTO ON**: muestra las variables a "tiempo real"
- **SIZE 23**: da más espacio para mostrar el servicio
- **M LI...**: te enseña los campos de una copy. Por ejemplo: M LI SALIDA OF COPY-...
- **F2**: ejecutar paso a paso (línea)
- **F4**: si le das sobre una variable, ves su valor
- **F6**: poner o quitar un punto de interrupción
- **F7/F8**: subes/bajas en la pantalla
- **F9**: ejecutas de punto a punto de interrupción
- **F10**: sale un monitor con todas las variables de una copy, útil cuando usas el comando "M LI"

## Conceptos avanzados

### Trabajos

Hay tres tipos de trabajos: por lotes (BATCH, el sistema lo asigna como JOB), interactivo (TSU) y Started task (STC). Un trabajo siempre tiene un flujo: datos de entrada, ejecuta u proceso (utility) y obtiene un resultado.

Un JOB esta formado por pasos o steps y cada uno ejecuta un programa o utility.

### Información sobre una partición

Si nos vamos a cualquier carpeta del HOST y ponemos una I (info) a la derecha nos aparecerá información útil.

```

Command - Enter "/" to select action ----- Message
i          SYS1.PARMLIB                      Info - I

Data Set Name . . . : SYS1.PARMLIB

General Data
Volume serial . . . : SYSBF0
Device type . . . : 3390
Organization . . . : PO = PARTICIONADO
Record format . . . : FB = BLOQUES FIJOS
Record length . . . : 80
Block size . . . : 27920
1st extent cylinders: 2
Secondary cylinders : 0
Data set encryption : NO

Current Allocation
Allocated cylinders : 2
Allocated extents . : 1
Maximum dir. blocks : 12

Current utilization
Used cylinders . . : 1
Used extents . . . : 1
Used dir. blocks . : 1
Number of members . : 4

Dates
Creation date . . . : 2008/09/17
Referenced date . . : 2023/08/23
Expiration date . . : ***None***

```

Crear un dataset con las mismas características:

```

Data Set Information

Command ==> =3.2_

Data Set Name . . . : SYS1.PARMLIB

General Data
Volume serial . . . : OS39M1
Device type . . . : 3390
Organization . . . : PO
Record format . . . : FB
Record length . . . : 80
Block size . . . : 6160
1st extent tracks . : 13
Secondary tracks . : 15

Current Allocation
Allocated tracks . . : 13
Allocated extents . : 1
Maximum dir. blocks : 225

Current Utilization
Used tracks . . . . : 13
Used extents . . . : 1
Used dir. blocks . . : 4
Number of members . : 73

Creation date . . . : 1997/03/27
Referenced date . . : 2022/08/30
Expiration date . . : ***None***

```

Nombre en comillas simples para que sea absoluto

```

Menu  RefList  Utilities  Help

Data Set Utility

Option ==> a

A Allocate new data set          C Catalog data set
R Rename entire data set        U Uncatalog data set
D Delete entire data set        S Short data set information
blank Data set information      V VSAM Utilities

ISPF Library:
Project . . . 
Group . . . 
Type . . . 

Other Partitioned, Sequential or VSAM Data Set:
Data Set Name . . . 'MC'
Volume Serial . . .      (If not cataloged, required for option "C")

Data Set Password . . .      (If password protected)

```

```

Data set allocated

```

hay que poner algo dentro para poder editarlo. Ir a otra carpeta y copiarlo

```

Menu  Options  View  Utilities  Compilers  Help

DSLIS - Data Sets Matching MCORNER.JCL          No members in data set
Command ==>                                     Scroll ==> CSR

Command - Enter "/" to select action          Message          Volume
-----
E          MCORNER.JCL                          OS39M1
***** End of Data Set list *****

```

## Sintaxis de JCL

Todas empiezan // y a partir de la columna 72 va a ser ignorado porque, aunque tengamos 80, los últimos se usa como CRC (línea bien puesta, desde los tiempos de las tarjetas perforadas)

Reglas a tener en cuenta:

1. Toda línea empieza con //
2. Se escribe desde la primera parte hasta el carácter 72
3. Empieza con JOB
4. Todo en mayúsculas
5. Mínimo un paso



6. Todo paso lleva una definición de datos (sentencia DD como en PASCAL)
7. Termina cuando se encuentra un // final

El nombre del JOB debe tener como máximo 8 caracteres y empezar con un carácter alfabético.

JOB = Trabajo

PROC = Proceso

### Inicio

```
//MIJOB JOB ACOUNTING,USUARIO,CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1),
```

```
//      NOTIFY=&SYSUID,REGION=256K
```

ACCOUNTING – para saber los recursos usados

USUARIO – quien lo usa

CLASS=A → iniciador que escuche por la clase A

MSGCLASS=A → en que clase de salida deja los mensajes del JOB

MSGLEVEL=(1,1) → como de hablador es el job, logs. (1,1) todos los mensajes del job y todos los recursos (del sistema)

NOTIFY=&SYSUID – nos diga si ha finalizado el job (bien o mal), a nuestro usuario (el que ha lanzado el JOB)

,REGION=256K todo el JCL va a ocupar 256k y puede que tengamos problemas de memoria.  
OM = infinito (cuidado que puede comerse toda la RAM)

### Paso

```
//PASO1 EXEC PGM=[nombre del programa],REGION=256K
```

IEBGENER = copiar lo que poner lo que hay en SYSUT1 en SYSUT2, sin parámetros adicionales. Necesita memoria para ser ejecutado

REGION=256K para cada paso

//SYSUT1 DD \* → todo lo que haya después del \* se adjunta al SYSUT1 hasta el próximo //. Si pones un DATA todo lo que haya entre eso y el /\* no va por el parser y puedes poner lo que quieras (incluso el //)

/\* → end of data

```
//SYSUT2 DD SYSOUT=A → lo deja impreso en la clase A
```

```
//SYSPRINT DD SYSOUT=A → que imprima la información como un log
```

//SYSIN DD DUMMY → parámetro de entrada. Cuando tiene DUMMY es que no tiene. Esto se debe a que siempre se debe poner

//

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT      MCORNER.JCL(MIJCL) - 01.00                      Columns 00001 00072
Command ==> _____ Scroll ==> CSR
***** ***** Top of Data *****
000100 //MIJOB   JOB MCORNER,LARRIETA,CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1),
000200 //          NOTIFY=&SYSUID
000300 //PAS01   EXEC PGM=IEBGENER,REGION=256K
000400 //SYSUT1   DD *
000500 Mi primer JCL dentro de mi entorno OS/390!!!
000600 //SYSUT2   DD SYSOUT=A
000700 //SYSPRINT DD SYSOUT=A
000800 //SYSIN     DD DUMMY
000900 //
***** ***** Bottom of Data *****

```

```

1 //MIJOB   JOB MCORNER,LARRIETA,CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1),
  //          NOTIFY=&SYSUID
  IEFC653I SUBSTITUTION JCL - MCORNER,LARRIETA,CLASS=A,MSGCLASS=A,MSGLEV
2 //PAS01   EXEC PGM=IEBGENER,REGION=256K
3 //SYSUT1   DD *
4 //////////////////////////////////////////////////
5 //SYSIN     DD *          GENERATED STATEMENT
6 //////////////////////////////////////////////////

```

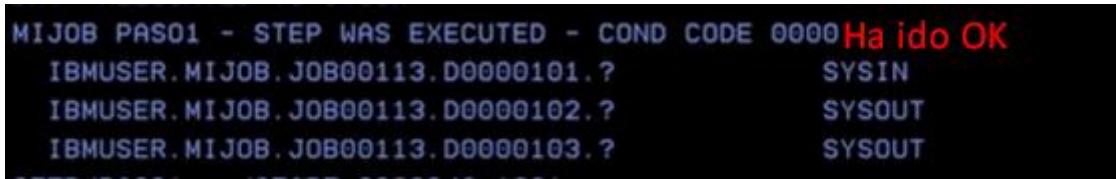
```

EDIT      MCORNER.JCL(MIJCL) - 01.01                      Columns 00001 00072
Command ==> _____ Scroll ==> CSR
***** ***** Top of Data *****
000100 //MIJOB   JOB MCORNER,LARRIETA,CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1),
000200 //          NOTIFY=&SYSUID
000300 //PAS01   EXEC PGM=IEBGENER,REGION=256K
000400 //SYSUT1   DD DATA
000410 //////////////////////////////////////////////////
000500 /    Mi primer JCL dentro de mi entorno OS/390!!!    /
000501 //////////////////////////////////////////////////
000510 /*
000600 //SYSUT2   DD SYSOUT=A
000700 //SYSPRINT DD SYSOUT=A
000800 //SYSIN     DD DUMMY
000900 //
***** ***** Bottom of Data *****

```

## Ejecución del JOB

Nos vamos a la barra de comando: SUB. El resultado está en 0.S.ST



```
MIJOB PAS01 - STEP WAS EXECUTED - COND CODE 0000 Ha ido OK
IBMUSER.MIJOB.JOB00113.D0000101.?      SYSIN
IBMUSER.MIJOB.JOB00113.D0000102.?      SYSOUT
IBMUSER.MIJOB.JOB00113.D0000103.?      SYSOUT
```

En el JOB del 0.S.ST si le pones un ? te limita y puedes ir seleccionado cada línea para ver los datos

## Procesos

Igual que el JOB, tiene una serie de pasos que se ejecuta una y otra vez, según lo necesitamos (día, mes, especiales, petición)

Tipos: De flujo (INSTREAM) y catalogado (CATALOG)

Pasos: declaración del PROC, ejecución y cambio de parámetros.

DSN substitution tiene como se llama de verdad el fichero tanto de entrada como salida