



# **CURSO DE GENERADOR**

## **(Versión 3.0)**

### **Índice**

- 1.- [Entorno Diagram9.](#)
  - 1.1.- [Estructura del básico en Servidor.](#)
  - 1.2.- [Estructura del básico en Cliente.](#)
  - 1.3.- [Estructura de las aplicaciones.](#)
- 2.- [Entorno DSGen.](#)
  - 2.1.- [Tipos de archivo.](#)
  - 2.2.- [Estructura de los menús de aplicaciones.](#)
- 3.- [Definición de tablas.](#)
  - 3.1.- [Campos.](#)
  - 3.2.- [Pantallas.](#)
  - 3.3.- [Índices.](#)
  - 3.4.- [Relaciones.](#)
  - 3.5.- [Programas.](#)
    - 3.5.1.- [Estructura.](#)
  - 3.6.- [Procesos.](#)
    - 3.6.1.- [Estructura.](#)
    - 3.6.2.- [Tipos.](#)
    - 3.6.3.- [Ficheros y campos.](#)
    - 3.6.4.- [Variables.](#)
      - 3.6.4.1.- [Ámbito.](#)
      - 3.6.4.2.- [Variables reservadas.](#)
      - 3.6.4.3.- [Variables persistentes.](#)
    - 3.6.5.- [Operaciones.](#)
      - 3.6.5.1.- [Numéricas.](#)
      - 3.6.5.2.- [Alfanuméricas.](#)

- 3.6.5.2.- [Con fechas.](#)
    - 3.6.6.- [Control de Flujo de programas.](#)
      - 3.6.6.1.- [Bifurcaciones condicionales.](#)
      - 3.6.6.2.- [Bifurcaciones incondicionales.](#)
      - 3.6.6.3.- [Bucles de instrucciones.](#)
      - 3.6.6.4.- [Bucles de ficheros.](#)
        - 3.6.6.4.1.- [Bucle declarado.](#)
        - 3.6.6.4.2.- [Bucle de lectura de ficheros de líneas.](#)
      - 3.6.6.5.- [Anidación de procesos.](#)
  - 3.7.- [Rutinas.](#)
  - 4.- [El Editor de textos del Generador.](#)
  - 5.- [Guía de Referencia.](#)

[A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)[\\_](#)

[ABORTA](#)

[ABRE](#)

[ABRET](#)

[ACABA](#)

[AFEGIR\\_FICHERO](#)

[ALARMA](#)

[ALFACALLDLL](#)

[AL\\_BUF](#)

[ATAR](#)

[ATRI](#)

[ATRI\\_DEF](#)

[AVISO](#)

[BLANCO](#)

[BLIN](#)

[BMENUG](#)

[BORRA](#)

[BORRA\\_HIJOS\\_PAN](#)

[BROWSE](#)

[BUCLE](#)

[BUCLELIN](#)

[CABEZA](#)  
[CAMPO ANCHO](#)  
[CAMPO ATRIBUTO](#)  
[CAMPO LINEAL](#)  
[CAMPO MODIFICA](#)  
[CAMPO POSICION](#)  
[CAMPO VISUAL](#)  
[CARGADLL](#)  
[CDEFECTO](#)  
[CIERRA](#)  
[CIERRAT](#)  
[CIMPR](#)  
[CLS](#)  
[COLOR](#)  
[COMPRIME](#)  
[CONFI](#)  
[CONSULTA CLAVE](#)  
[CONSULTA F CLAVE](#)  
[COPIA FICHERO](#)  
[CORRE](#)  
[CUADRO](#)  
[CUADRO D](#)  
[CUADRO S](#)  
[DAREGN](#)  
[DBASE](#)  
[DBASS](#)  
[DBIN](#)  
[DDEF](#)  
[DDEFECTO](#)  
[DEBUG](#)  
[DEFECTO](#)  
[DELINDEX](#)  
[DEL BUF](#)  
[DESCARGADLL](#)  
[DESCOMPRIME](#)

DESTRUYECONTROL

DETAR

DFICB

DFICE

DIMENSIONA

DIRECTORIO

DPAN

DSCORREO (DS\_CORREO\_ENVIA DS\_CORREO\_RECIBE)

EDITA

ENCAMPO

ENCLAVE

ENDATOS

ENTCOLUMNA

ENTLINEAL

ENVIA\_FICHERO

ERROR

ERROR10

ESPECIFICA

ESTADOCOM

ET

EXTERNOEXE

FABRE

FBORRA

FCIERRA

FGRABA

FINCOM

FINDIM

FINIMP

FININF

FINSI

FINTIMER

FINVENTANA

FLEE

FNOM

F\_TEXTO

GRABA  
GRABABMPPANTALLA  
GRABADEFSPANTALLAS  
GRABAMENUPANTALLAS  
GRABA\_NUMERO  
GRABACOM  
GRABAENDEF  
GRABASECUENCIAL  
GRABAVE  
GRAF  
HACIENDO  
HAZ  
HORA  
HACIENDO  
HAZ  
HORA  
INICIOCOM  
INCLUYE  
IMPRE  
IMPRIME  
INFOR  
IP\_REMOTE  
LEE  
LEECOM  
LEECOMBUFFER  
LEENDEF  
LEE\_NUMERO  
LEESECUENCIAL  
LEETECLA  
LETRA  
LIBERA  
LINEAINFORME  
LINEAL\_SIMPLE  
MANTE  
MENAV

[MENSA](#)  
[MENU](#)  
[MENUG](#)  
[MENUIITEM](#)  
[MODO\\_BMP](#)  
[MODO\\_RELACION](#)  
[MODULO](#)  
[MKDIR](#)  
[NOALARMA](#)  
[NOMAP](#)  
[NOMBRE\\_IP](#)  
[NOME\\_DAT](#)  
[NOPARA](#)  
[ONTIMER](#)  
[PARA](#)  
[PARTE\\_FICHERO](#)  
[PATH\\_DAT](#)  
[PATH\\_PAN](#)  
[PAUSA](#)  
[PDEFECTO](#)  
[PIEINF](#)  
[PINDA](#)  
[PINPA](#)  
[PINTA](#)  
[PINVE](#)  
[PONREGN](#)  
[POPV](#)  
[PRINT](#)  
[PRINTA\\_PAN](#)  
[PROCESO](#)  
[PTEC](#)  
[PULSATECLA](#)  
[PUSHV](#)  
[QBF](#)  
[QBT](#)

[QUE SISTEMA](#)  
[RATAR](#)  
[RBASS](#)  
[RCOMPRIME](#)  
[RCOPIA FICHERO](#)  
[RDEFECTO](#)  
[RDELTREE](#)  
[RDESCOMPRIME](#)  
[RDETAR](#)  
[REFRESCACONTROL](#)  
[RENOMBRA FICHERO](#)  
[REPINTA TEXTO](#)  
[RFABRE](#)  
[RFBORRA](#)  
[RFCIERRA](#)  
[RFGRABA](#)  
[RFLEE](#)  
[RGRABA NUMERO](#)  
[RLEE NUMERO](#)  
[RMDIR](#)  
[RMKDIR](#)  
[RRENOMBRA FICHERO](#)  
[RRMDIR](#)  
[RSUMA FICHERO](#)  
[RSYSTEM](#)  
[RTEMPO](#)  
[R\\_PDIR](#)  
[R\\_SDIR](#)  
[SAL](#)  
[SALTO DE LINEAS](#)  
[SCROLL](#)  
[SELECT](#)  
[SI](#)  
[SIALARMA](#)  
[SIGUE](#)

SINO  
SIPARA  
SLEEP  
SQL  
SUB\_EJECUTA  
SUB\_EJECUTA\_NP  
SUB\_PINPA  
SYSTEM  
TECLA\_FUNCION  
TEMPO  
VENTANA  
VERSIONRTME  
VETE  
WORD\_ABRE  
WORD\_ASIGNA\_TEXTO  
WORD\_CARGA\_TEXTO  
WORD\_IMPRIME  
WORD\_PROPIEDADES  
WORD\_SALVA  
XABRE  
XCIERRA  
XLEE  
XGRABA  
XLEE\_NUMERO  
XGRABA\_NUMERO  
XLEEDEDEF  
XGRABAADEF  
XPOSICION  
\_PDIR  
\_SDIR

Recogida de datos de un def.

Uso de OCX.

Manejo de ficheros Excel.

6.- Normas para el bloqueo de Registros.



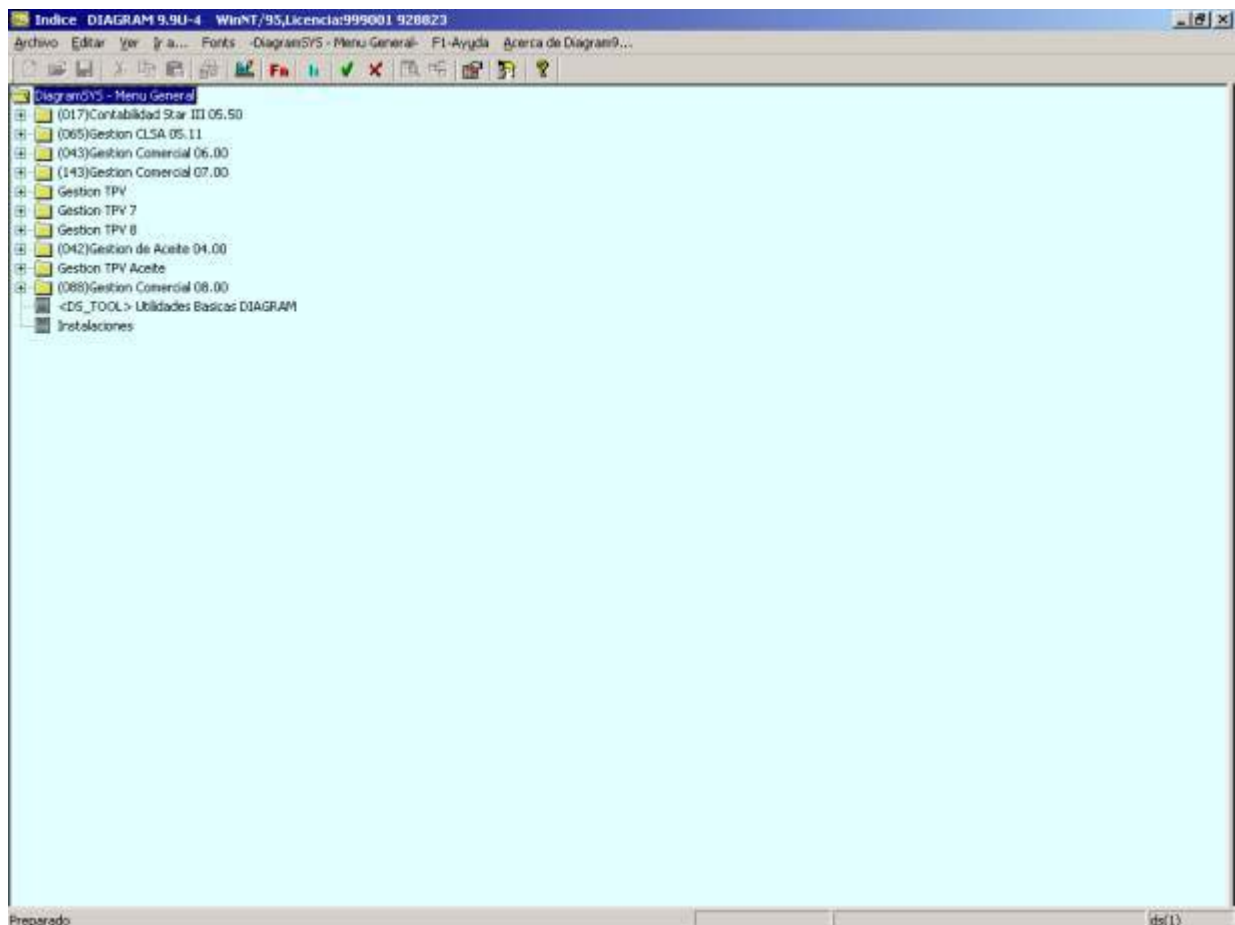
## 7.- [Prácticas.](#)

- 7.1.- [Tablas únicas.](#)
- 7.2.- [Tablas Cabecera-líneas.](#)
- 7.3.- [Tablas relacionadas.](#)
- 7.4.- [Cálculos y Procesos.](#)
- 7.5.- [Rutinas.](#)
- 7.6.- [Programas.](#)
- 7.7.- [Bucles.](#)

## 8.- [Guía de Errores.](#)

- 8.1.- [Lista de avisos del generador.](#)
- 8.2.- [Errores de tratamiento de ficheros C-ISAM](#)
- 8.3.- [Errores no usuales en instalaciones. Errores de desarrollo.](#)

## 1.- Entorno Diagram9.

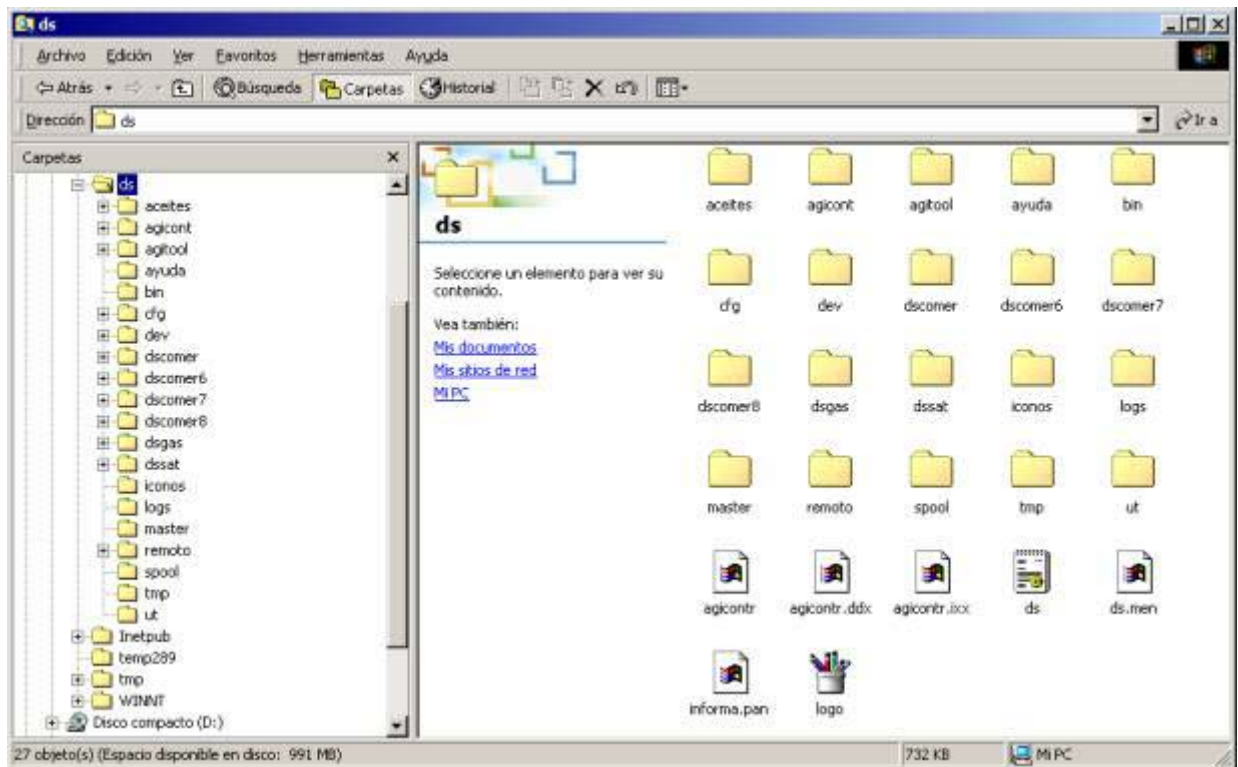


Básicamente, el entorno donde se ejecutan las aplicaciones de Diagram Software, S.L., está dividido en cinco zonas determinadas:

- Menú. -> Es la parte superior de la pantalla, y en él nos encontramos distintas opciones:
- Archivo -> Como en la mayoría de las aplicaciones Windows, el primer menú se llama archivo. De éste menú, sólo vamos a destacar tres opciones:
- Impresión sobre Excel. En caso de que tengamos instalado Microsoft Excel, y activemos esta opción (se activa y desactiva haciendo **clíc** sobre ella), cualquier impresión que efectuemos desde las aplicaciones Diagram se encasillará en las celdas de una hoja de cálculo nueva.
- Impresión Standard. De igual activación que la opción anterior, si esta opción está seleccionada, la impresión que efectuemos se dirigirá a las impresoras que tengamos instaladas en el entorno Windows.

- Salir. A diferencia de otras aplicaciones Windows, al pulsar sobre esta opción, únicamente saldremos de la pantalla que actualmente tengamos abierta. Sólo saldremos del entorno Diagram9 si es que estamos en su menú principal.
- Editar.
- Ver.
- Ir a.
- Fonts.
- –DIAGRAM SYS- Menú General.
- En esta opción se desglosarán los sucesivos menús de las aplicaciones en las que entremos.
- Barra de botones.
- Árbol de menús. Es la ventana que está en la parte de la izquierda de la aplicación con aspecto de explorador de Windows. En este menú se ven las aplicaciones instaladas. Se puede recorrer con las teclas *flecha arriba* y *flecha abajo*. Para desplegar carpetas se puede utilizar la *flecha derecha* y para plegarlas se puede utilizar la *flecha izquierda*. Para entrar en una opción se puede utilizar *Intro*. Una vez escogida una opción, ésta sustituirá el logotipo de Diagram Software, S.L. que aparece en la pantalla de aplicaciones.
- Pantalla de aplicaciones. En esta ventana es dónde se van a ejecutar las aplicaciones de Diagram.
- Barra de estado. Es la barra que aparece en la parte inferior del entorno y podrá contener información como el nombre de usuario, número de registro actual (en un bucle o búsqueda), etc.

### **1.1.- Estructura del básico en Servidor.**



La estructura de directorios (carpetas) en el servidor es la siguiente:

Directorio básico:

***Unidad:\ds*** (caso de Windows)

Ó

***/u/ds*** (caso de Unix)

Este directorio puede tener otro nombre, dependiendo de cómo se haya hecho la instalación, pero los directorios que cuelgan de él no pueden y son los siguientes:

**Ayuda**

En este directorio se guarda la ayuda referente a Diagram 9.

**Bin**

Aquí se guardan los archivos necesarios para la ejecución del servidor, tales como *diagram9.exe*, *serverds.exe*, *rwnetcon.exe*, y otros archivos auxiliares.

**Cfg**

**Dev**

En este directorio se guarda las configuraciones de dispositivos vinculados al entorno, tales como *impresoras, usuarios, terminales, etc*, y un archivo muy importante que se llama ***ds.reg***. En este archivo se guarda la información necesaria para que Diagram9 configure el menú principal dependiendo de las aplicaciones que tenga instaladas y con permiso de ejecución.

Iconos

Logs

Aquí se guardan los registros de acciones realizadas por los usuarios en cada uno de los ficheros de datos del programa.

Master

En esta carpeta se guardan los archivos de instalación comprimidos para efectuar una eventual reinstalación.

Remoto

Spool

Como su propio nombre indica, es dónde se realizan las colas de impresión.

Tmp

Carpeta para archivos temporales.

Ut

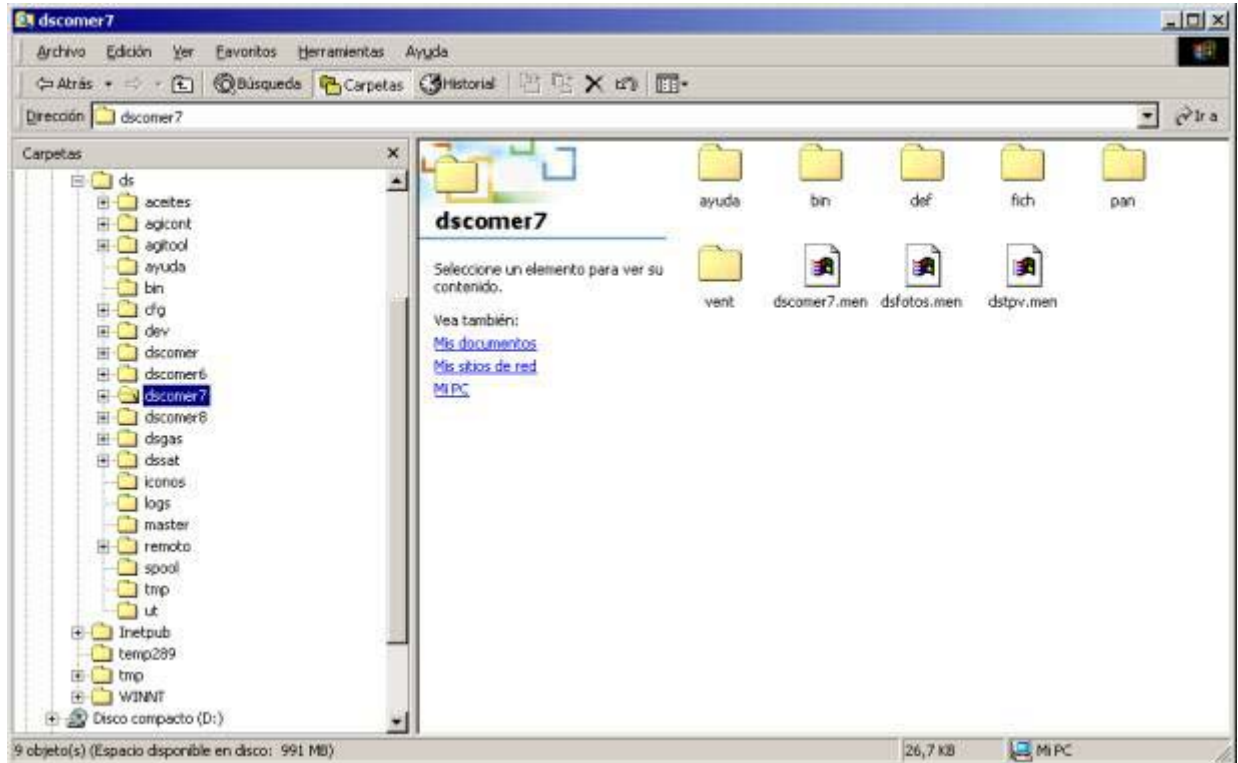
Aquí se guardan archivos de utilidades como por ejemplo el *aginfont* o *aginfobx*, que sirve para generar y modificar los informes.

Además de estos directorios, habrá una entrada para cada una de las aplicaciones instaladas en el servidor.

## **1.2.- Estructura del básico en Cliente.**

La estructura del básico en el cliente es prácticamente la misma que en el servidor, con la diferencia fundamental que no hay entradas para las aplicaciones.

### 1.3.- Estructura de las aplicaciones.



Las aplicaciones Diagram tienen una estructura idéntica:

*/ayuda*

Aquí se guardan los ficheros de ayudas de los distintos subprogramas.

*/bin*

Aquí se guardan los ejecutables (no es que se puedan ejecutar directamente, dependen del run-time), de las aplicaciones.

*/def*

Aquí se guardan las estructuras de los ficheros de datos que van a utilizar las aplicaciones.

*/fich*

Si la aplicación aquí instalada no soporta multi-empresa, aquí se guardan los ficheros de datos. Si, por el contrario, sí que lo soporta, se guarda un directorio para cada una de las empresas. Además se

incluye en este directorio un archivo con la extensión *.dir*, que contiene una lista de los ficheros que existen en el directorio *def*, y que se utiliza para algunas funciones. También está en este directorio el fichero con los datos de las empresas.

*/pan*

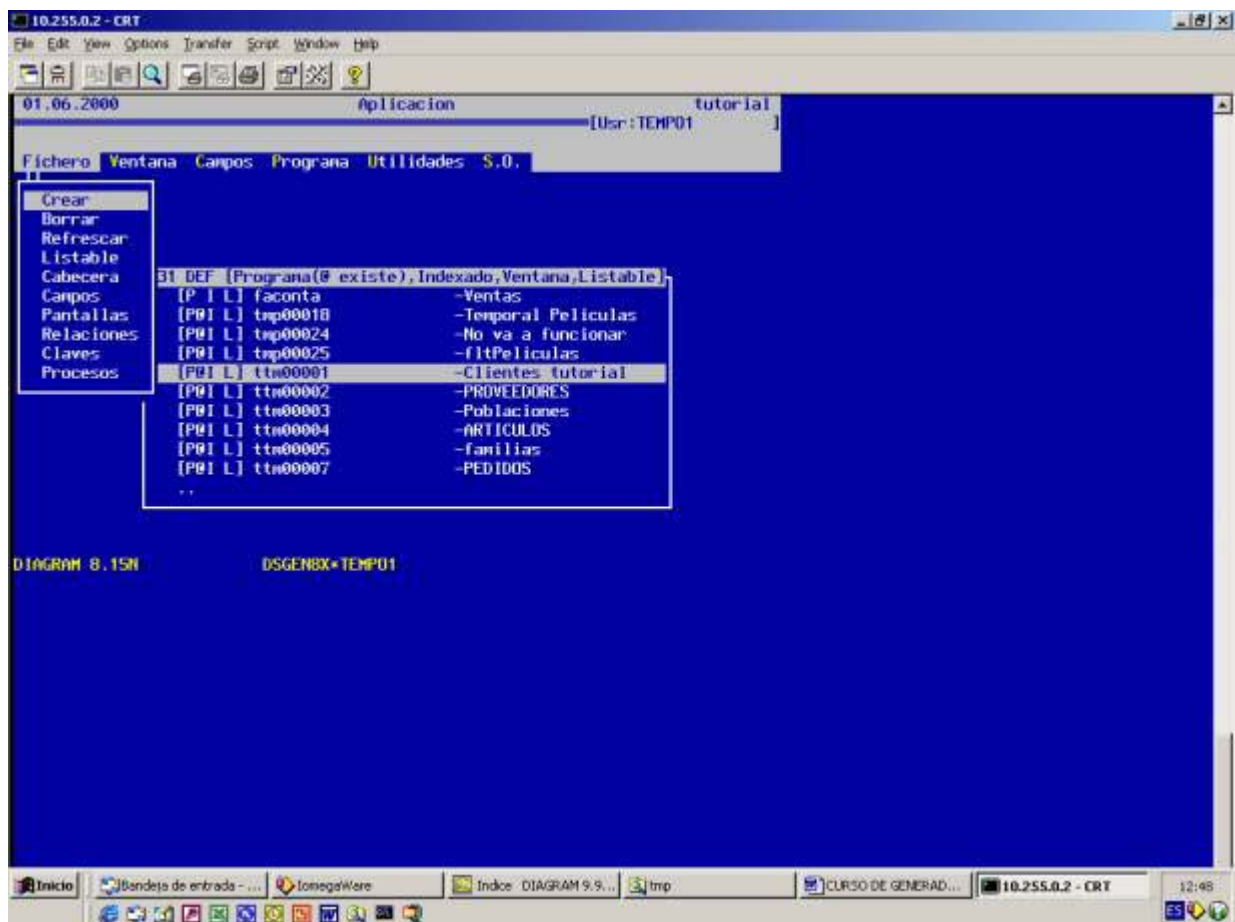
En esta carpeta se guardan las pantallas que utilizan los distintos subprogramas para visualizar datos.

*/vent*

*\*.men*

Estos archivos son los menús propios de la aplicación.

## 2.- Entorno DSGen.



Actualmente, el entorno de programación de Diagram es en modo texto, aunque ya se está trabajando en su formato gráfico.

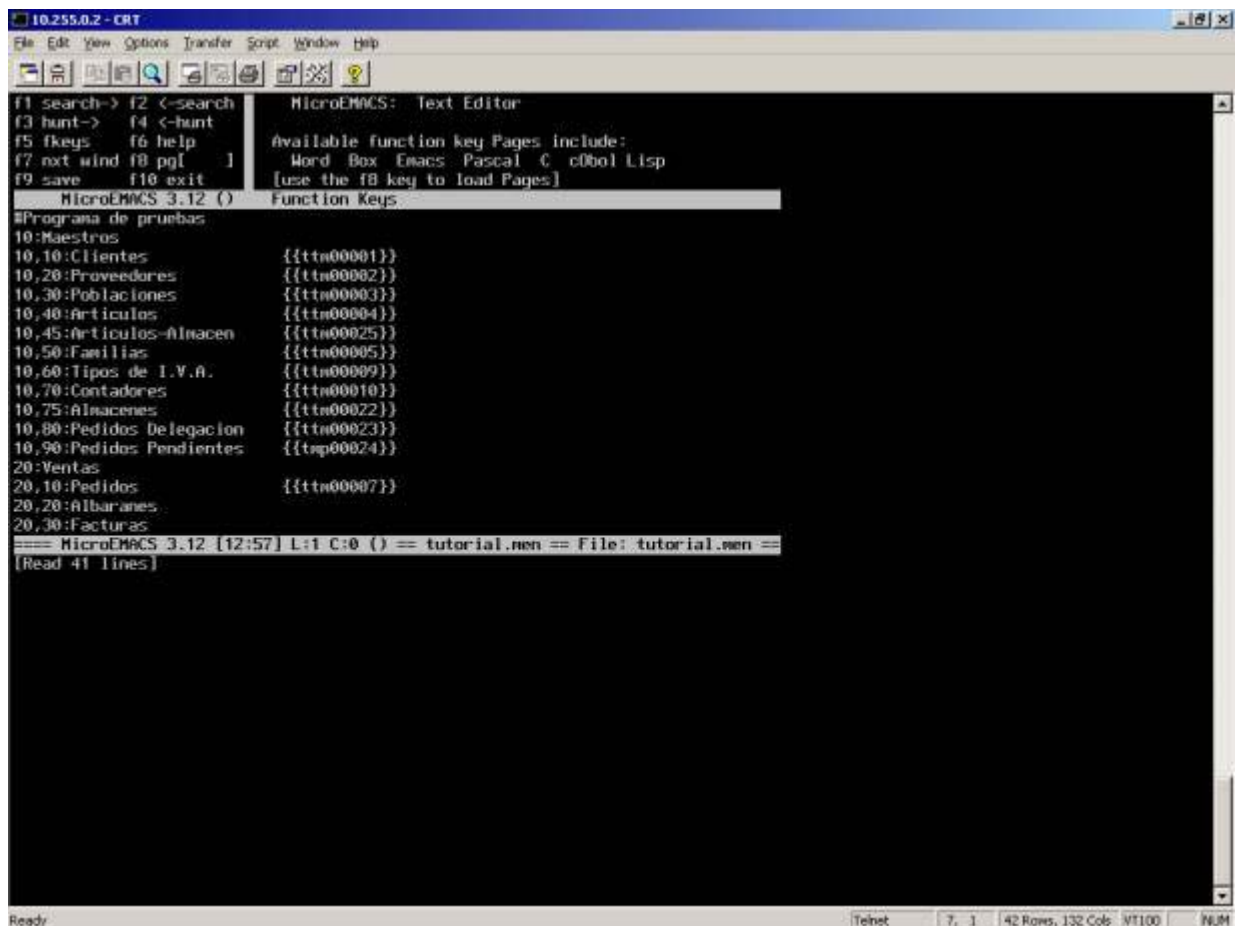
Consta de varios menús. El desplazamiento entre opciones es con las *flechas* del teclado y la selección de opciones es con la tecla *intro*.

## 2.1.- Tipos de archivo.

Con el generador creamos distintos tipos de archivo, que se guardan en directorios distintos:

/pan → *.pan	(archivos de definición de pantalla)
/def → *.mas	(archivos de definición de estructuras de fichero)
*.inf	(archivos de definición de informes)
/cal → *.cal	(archivos de procesos)
/inc → *.mas	(archivos de estructuras incluidas)
/vent → *.mas	(archivos de ventana)

## 2.2.- Estructura de los menús de aplicaciones.



```
10.255.0.2 - CRT
File Edit View Options Transfer Script Window Help
MicroEMACS: Text Editor
Available function key Pages include:
Word Box Emacs Pascal C cObol Lisp
[use the f8 key to load Pages]
MicroEMACS 3.12 ()
Function Keys
#Programa de pruebas
10:Maestros                {{{tn00001}}}
10,10:Clientes             {{{tn00002}}}
10,20:Proveedores         {{{tn00003}}}
10,30:Poblaciones         {{{tn00004}}}
10,40:Articulos           {{{tn00025}}}
10,45:Articulos-Almacen   {{{tn00005}}}
10,50:Familias            {{{tn00009}}}
10,60:Tipos de I.V.A.     {{{tn00010}}}
10,70:Contadores          {{{tn00022}}}
10,75:Almacenes           {{{tn00023}}}
10,80:Pedidos Delegacion  {{{tp00024}}}
10,90:Pedidos Pendientes
20:Ventas                 {{{tn00007}}}
20,10:Pedidos
20,20:Albaranes
20,30:Facturas
==== MicroEMACS 3.12 [12:57] L:1 C:0 () == tutorial.men == File: tutorial.men ==
[Read 41 lines]
```

La estructura de un menú de aplicación es sencilla. La primera línea contiene el título del menú, precedido de un #.

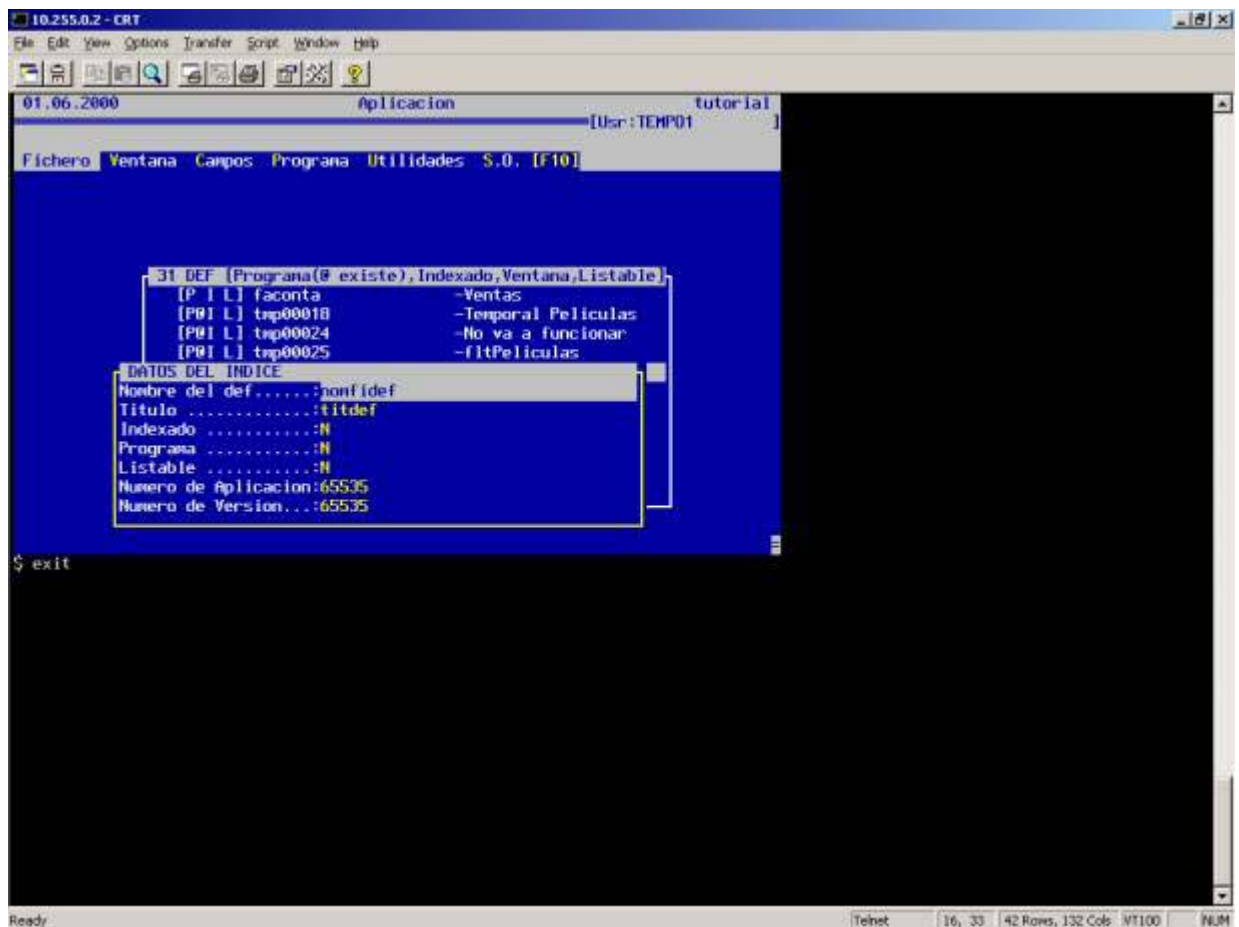
Las opciones se detallan a continuación precedidas de un número. Si añadimos una coma y otro número, se creará una rama en el árbol de menús. Tras los dos puntos aparece el título de la opción en el menú. Si esta opción



tiene un programa, es decir, no es la cabecera de una rama, el nombre de esta opción se escribirá entre símbolos {{ opción }}.

### 3.- Definición de tablas.

Para definir tablas, entraremos en el generador y pulsaremos *intro* en la opción *Fichero*. Se desplegará un menú. Lo primero que escogeremos es la opción *Crear*.



En *nombre del DEF* escribiremos un texto de 8 caracteres que será el nombre para esta tabla.

En *Titulo*, escribiremos una descripción del uso u objetivo de la tabla.

En *Indexado*, indicaremos si esta tabla tiene o no índices.

En *Programa*, indicaremos si esta tabla tiene o no un programa asociado.

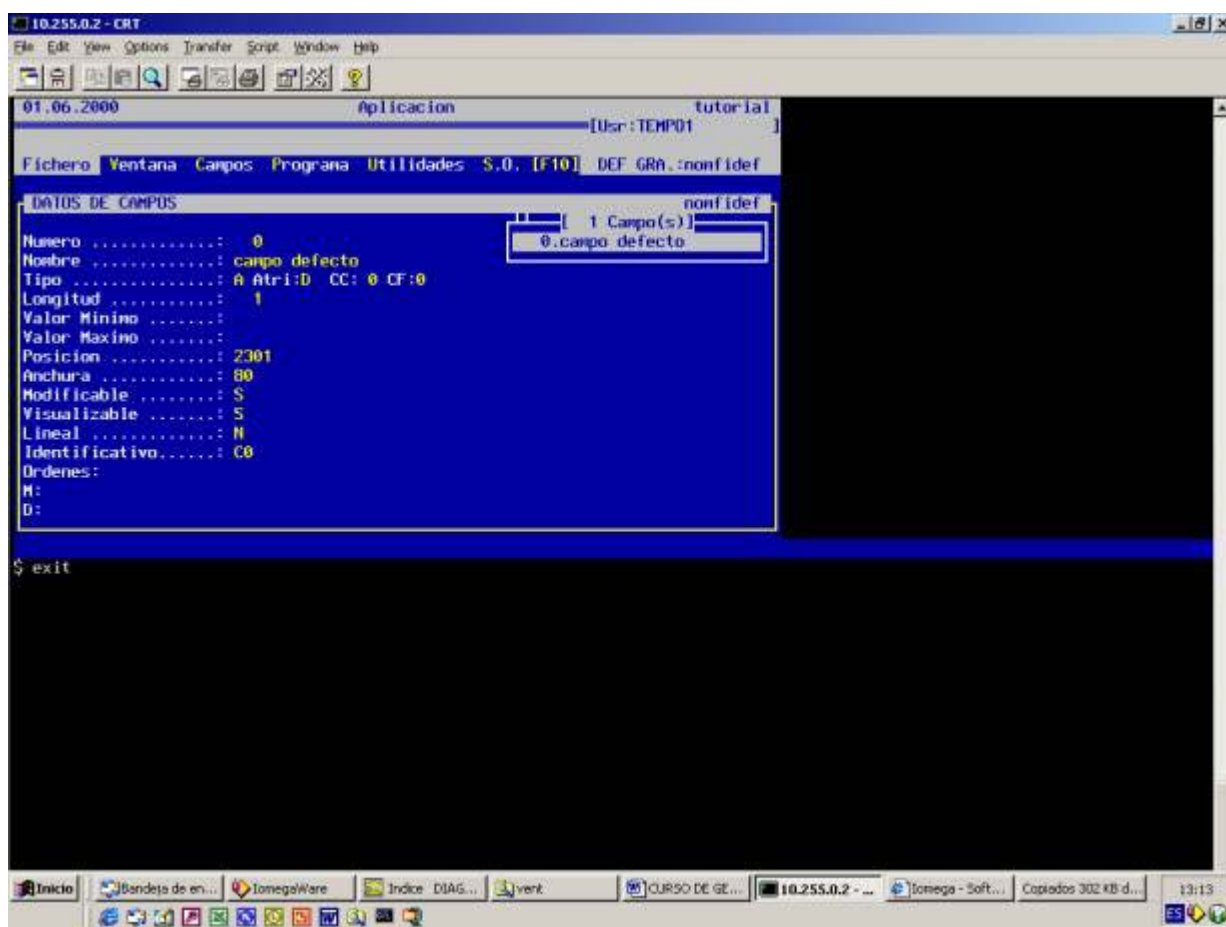
En *Listable*, diremos S si es que esta tabla tiene que salir en la lista de archivos listables.

El número de aplicación y versión son controles que utilizará el básico para detectar si el cliente tiene o no permiso de ejecución de esta tabla.

### 3.1.- Campos.

Para añadir campos a nuestra tabla, escogeremos de nuevo la opción *Fichero* y después la opción *Campos*.

Se desplegará una pantalla como la que sigue:



Si pulsamos *intro*, nos permitirá entrar en la definición del primer campo (usualmente la clave primaria). Podemos desplazarnos entre los datos de definición de los campos con las flechas *arriba* y *abajo*.

Una vez introducidos los datos del primer campo, para añadir otros, pulsaremos la tecla *Ins*, y para borrarlos la tecla *Supr*. ¡Ojo! Borrar campos es

peligroso, puede desestabilizar la estructura fichero-pantalla. Recomendamos, en la medida de lo posible, no borrar.

Los datos que nos pide en cada campo son los siguientes:

*Nombre:* Evidentemente, es el nombre del campo.

*Tipo:* Tipo de datos que contendrá:

A     Alfanumérico

N     Numérico

F     Fecha

*Longitud:* Si es un dato de tipo A o N, diremos el número de caracteres que va a medir como máximo.

*Valor mínimo:* Si es un dato de tipo numérico, indicaremos el valor mínimo que deberá tener el campo (p.ej. – 99999999.999). Si es un dato de tipo alfanumérico de tamaño 1, podremos indicar los valores que puede aceptar (p.ej. SNsn).

*Valor máximo:* Si es un dato de tipo numérico, indicaremos el valor máximo que deberá tener el campo (p.ej. 99999999.999).

*Posición:* Este valor no debemos modificarlo. Se actualiza automáticamente al generar la pantalla.

*Anchura:* Este valor si está en 80, tomará el valor máximo disponible para el campo. Si le ponemos un valor inferior al tamaño del campo, el básico realizará un scroll en el campo.

*Modificable:* Si vamos a permitir la entrada en este campo para su modificación indicaremos S, en caso contrario N.

*Visualizable:* Si este campo se puede visualizar en pantalla, indicaremos S.

*Lineal:* Si este campo es de tipo lineal (por ejemplo los campos de las líneas de una factura), indicaremos S.

*Identificación:* No usar.

**Órdenes:** Aquí es dónde diremos los procesos que tiene asociados el campo. Además de los procesos, podemos indicarle:

- En caso de que haya una o varias relaciones a otro fichero, un dato de ese otro fichero que escribirá automáticamente en el campo actual. Por ejemplo, tenemos el campo 4 que es el código de la Forma de Pago (fichero ttma0100) de un cliente, y en el campo 5 tenemos la Descripción de la misma. Hemos creado una relación del campo 4 al fichero de formas de pago. Bien pues, si en órdenes ponemos D002ttma0100, cuando escojamos una forma de pago en el formulario, nos escribirá directamente el campo 1 (recordar que siempre se pone el campo +1 en las órdenes) del fichero de formas de pago en nuestro campo 5.
- En caso de que haya más de una relación con el fichero cuyo valor por defecto queremos escribir en nuestro fichero, en cada orden en que pongamos valor por defecto añadiremos al final un número ordinal referente al número de orden de la relación con el susodicho fichero. Es decir, para la primera relación escribiremos D002ttma01001, para la segunda D002ttma01002, para la tercera D002ttma01003, y así sucesivamente.
- En caso de que tengamos un fichero con la cabecera (por ejemplo de facturas) y otro de líneas, el campo número de factura no queremos que se modifique, si que se vea pero, sin embargo, queremos que se actualice sólo al dar de alta una ficha. Lo que haremos será poner el campo (en el fichero de líneas) como *No visualizable*, *No Modificable*, *No Lineal*, y en órdenes le pondremos (suponiendo que el fichero de cabecera es el ttma0200, y el campo número de factura es el 0) D001ttma0200.
- En caso de que no queramos guardar el valor del fichero relacionado sino que tan sólo queremos visualizarlo, le indicaremos al DEF en órdenes, la fila y columna de pantalla en que lo haremos. Es decir, para el primer ejemplo anterior, si en lugar de querer guardar el valor de la descripción de la forma de pago, queremos visualizarla en la posición *fila 14*, *columna 25*, escribiremos P0001425002ttma0100. De igual modo

añadiremos 1, 2, 3, etc. en caso de que haya más de una relación al fichero.

*M:* El texto que pongamos en este campo aparecerá como mensaje al entrar en este campo y desaparecerá al salir del campo.

*D:* En este dato pondremos el valor por defecto que tomará el campo.

Cuando acabemos de introducir todos los campos, pulsaremos *ESC* o (*ESC en UNIX*) para salir y guardar cambios.

### **3.2.- Pantallas.**

Seguidamente, podemos entrar a crear la pantalla de mantenimiento de la tabla. Aparecerá una ventana como la que sigue. Lo primero que tendremos que escribir es su nombre (habitualmente el mismo nombre que el de la tabla o "DEF"). El siguiente dato que introduciremos serán las coordenadas que ocupará dicha pantalla, teniendo en cuenta que empiezan en 0500 (fila 5 columna 0) y acaban en 2380 (fila 23 columna 80).

Una vez introducidos estos datos, el generador pondrá la extensión *.pan* a nuestro fichero y podremos pulsar *ESC* para entrar en la pantalla.

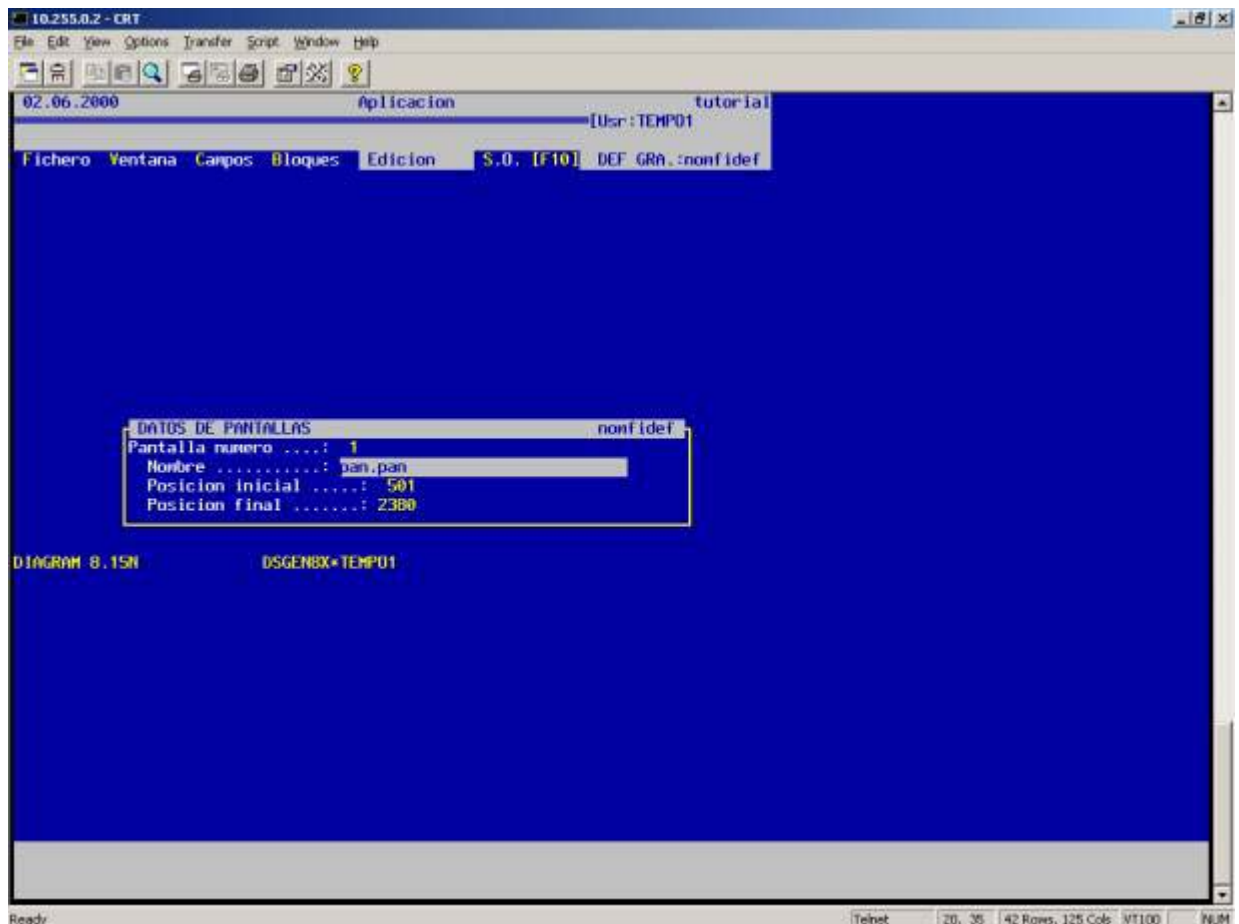


DIAGRAM 10: En la futura versión del run-time, si el def tiene varias pantallas y después del nombre de la primera pantalla ponemos “,” (coma pipe), la segunda pantalla se colocará inmediatamente debajo de la primera. Si por el contrario ponemos en cualquier pantalla después de su nombre “,nombrepestaña” nos pondrá en pestañas tantas pantallas como pongamos y en el título de la pestaña nos pondrá nuestro texto.

Una vez entremos en la pantalla, podemos escribir los textos que queramos que aparezcan (p.ej. Nombre: ), y pulsando la tecla F6 nos aparecerá una lista de campos para incluirlos en pantalla.

Pulsando F7, yendo a la esquina contraria y pulsando F8 marcamos un bloque. Una vez marcado podemos:

- Hacer un cuadro.
- Mover el bloque.
- Copiar el bloque.
- Eliminar el bloque.

Si pulsamos *AvPag* y después F9, nos refrescará el contenido de la pantalla.

Por último, si pulsamos F10, escogemos la opción *Atributos* y después la opción *Anular Colores*, la visualización de la pantalla en Windows será mejor.

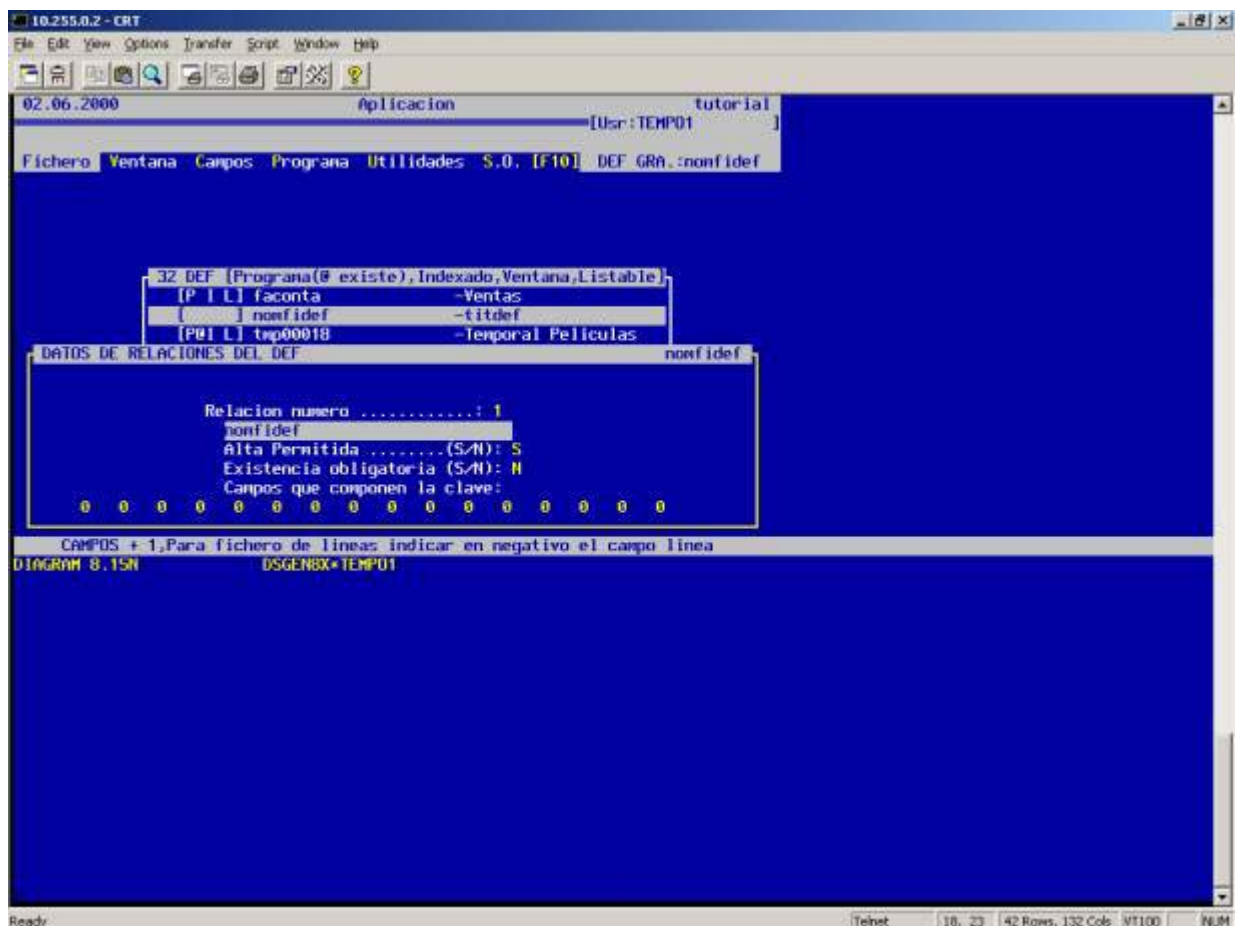
### 3.3.- Índices.

Para crear índices a la tabla, tenemos que entrar en la opción *Claves*.

En esta opción diremos el número de campos que componen la clave y su número en el fichero teniendo en cuenta que tenemos que poner el número +1. Por ejemplo, si el campo es el 0 (cero) pondremos que es el campo 1 (uno).

### 3.4.- Relaciones.

Para añadir relaciones entre esta tabla y otras, entraremos en la opción *Relaciones*.



En el nombre que nos pide, escribiremos el nombre del fichero con el cual tenemos un campo relacionado en la tabla corriente.

Si contestamos que *S* a *Alta permitida*, cuando introduzcamos un dato en el campo y el básico no encuentre un dato idéntico en la tabla relacionada, nos sugerirá automáticamente que lo demos de alta.

Si contestamos que *S* a *Existencia obligatoria*, el básico nos exigirá que exista un dato idéntico en la tabla relacionada.

Por último, indicaremos el (o los) campo relacionados en la tabla. En caso de la tabla relacionada sea un fichero de líneas de la actual (por ejemplo si nuestra tabla fuera la cabecera de los albaranes y quisiéramos indicar la relación con la tabla de líneas), el número de campo que indicaremos será el que contenga el número de línea de la tabla pero en negativo, es decir, si la tabla líneas tiene estos campos:

0	ALBARAN	N	6
1	LINEA	N	3

La relación con el fichero de líneas haría referencia al campo  $-2$  (recordar el número de campo + 1).

### **3.5.- Programas.**

Para crear el ejecutable y el mantenimiento automático hay que entrar en la opción *Programa* y hacer que la tabla actual sea *Generable*. A continuación volver a entrar en la opción *Programa* y decirle *Generar*. Una vez hecho esto, ya se puede comprobar que el programa funcione.

**NOTA: HACER LAS PRÁCTICAS 1, 2, 3 y 4.**

#### **3.5.1.- Estructura.**

Cuando generamos una tabla, se crea un mantenimiento (*alta, consulta, modificación, baja*) automáticamente. No obstante, es muy posible que queramos realizar nosotros mismos el mantenimiento, bien porque no sea una ficha para dar bajas ni modificaciones, bien por cualquier otro motivo.

En esos casos, tendremos que escribir un proceso en la opción *Procesos*. En esta opción podemos añadir varios procesos (con la tecla *Ins*), o suprimir procesos (con la tecla *Supr*), pero si tan sólo hay un proceso (lo más habitual), recomendamos llamarlo igual que al Def.

La estructura que debe seguir un programa es la siguiente:



(Notar que todas las órdenes comienzan con un pipeline '|' y terminan con un punto y coma ';')

|FICHEROS;

{Este grupo tenemos que incluirlo siempre, en él indicaremos los ficheros que vamos a utilizar en el programa, como mínimo el fichero en que nos encontramos}

ttma0001 #0; || Provincias

{En la línea anterior, hemos incluido el nombre de fichero a utilizar y el código que va a tener para el programa, siempre precedido de una parrilla '#'. Notar que tras el punto y coma hemos puesto dos pipelines '|', esto son comentarios. El generador dejará de leer a partir de ellos. También podemos utilizar la estructura de comentarios que tiene C, comenzando al principio de una línea con /\* el generador ignorará lo que haya escrito hasta que se encuentre al principio de otra línea \*/}

|FIN;

|VARIABLES;

{El grupo *variables*, sólo es necesario ponerlo si realmente vamos a utilizar alguna variable en nuestro programa}

|FIN;

|CAMPOS;

{El grupo *campos*, sólo es necesario ponerlo si realmente vamos a utilizar alguna variable en nuestro programa}

|FIN;

|PROGRAMA;

{El comienzo del código del programa se indica con la sentencia *PROGRAMA*. Todo lo que haya entre ésta y *FPRO* son instrucciones que se ejecutarán secuencialmente. El programa acabará cuando se encuentre ésta última}

|FPRO;

### **3.6.- Procesos o cálculos.**

No hay diferencias apreciables entre un cálculo y un proceso, ni en sintaxis, ni en utilización.

Definimos un proceso como la serie de operaciones e instrucciones con las cuales manipulamos las bases de datos definidas con DSGEN, aparte de los simples mantenimientos o como complemento de éstos. El lenguaje en el que está implementado es un lenguaje propio de tipo estructurado y en castellano. Cada proceso puede ejecutarse independientemente del resto, siempre que no exista un enlace entre ellos, por ello la diferencia fundamental entre el DS y los lenguajes de alto nivel, más conocidos, es no poseer necesariamente un cuerpo principal desde donde se invocan a todas las funciones presentes, aunque se puede simular en caso de requerirse. Por lo demás, presta la suficiente potencia de cálculo aritmético financiero y de manejo de pequeños campos de información como para considerársele un lenguaje completo y eficaz.

Todos los procesos relacionados entre sí se agrupan en un fichero sin extensión y que deberá almacenarse en un directorio denominado CAL, integrado en el resto de directorios que conformarán la aplicación que vamos a generar.

Un fichero de procesos consta de los siguientes bloques básicos:

- Declaración de ficheros.
- Declaración de variables.
- Declaración de campos.
- Procesos.

Todas las instrucciones deben finalizar con el símbolo ";" y, aunque normalmente se escriben de forma secuencial, es posible agrupar varias en una misma línea. Algunas de ellas serán operaciones matemáticas y otras ejecutarán determinadas sentencias, con o sin parámetros, que deberán comenzar siempre por el símbolo "|", código ASCII 124 también denominado "pipe". Por ejemplo:

suma = x + y;

Operación matemática

|[ABRE](#) #0;

Sentencia con parámetro (#0)

|[PAUSA](#);

Sentencia sin parámetros.

Los procesos son subrutinas que se ejecutarán en el campo desde el cuál sean llamadas según el tipo de evento que tengan definido, o bien desde una instrucción directa de ejecución desde otro proceso.

La manera de invocarlos en un campo es añadir en la definición de la tabla, en el campo *Ordenes* la letra E mayúscula seguida del nombre del proceso. ¡Atención!: El generador distingue entre mayúsculas y minúsculas.

La manera de invocarlos desde otro proceso es dando el comando

|[HAZ](#) *nombreproceso*;

Aunque en este caso y, puesto que no podemos aprovechar el tipo de proceso, es más interesante utilizar las *RUTINAS*, puesto que éstas no están limitadas en número y son capaces de hacer casi lo mismo.

### 3.6.1.- Estructura.

La estructura de los procesos es sencilla:

|[PROCESO](#) *nombreproceso*; |TIPO *numerodetipodeproceso*;

...

{Aquí incluiremos las órdenes}

|FPRC;

### 3.6.2.- Tipos.

Según el momento o la forma en que deba ejecutarse, hay distintos tipos de proceso:

- 0: Este proceso se ejecuta cuando se sale del campo en el que está definido. No es necesario ponerlo, pues si no especificamos el tipo en la definición del proceso, el generador asume que es de tipo 0.
- 1: Se ejecuta antes de todo al entrar en una ficha en modo modificación o en modo baja.
- 2: Este proceso permite que ciertas operaciones aritméticas se ejecuten en su modo normal si estamos en un alta, o de modo inverso si estamos en modo modificación o baja.

Para indicar que queremos que una operación sea reversible, tendremos que poner un punto tras ella, es decir:

‘+.’ Operará como ‘+’ en alta y como ‘-’ en modificación o baja.

‘-.’ Operará como ‘-’ en alta y como ‘+’ en modificación o baja.

‘\*.’ Operará como ‘\*’ en alta y como ‘/’ en modificación o baja.

‘/.’ Operará como ‘/’ en alta y como ‘\*’ en modificación o baja.

Esta clase de procesos se ejecuta al entrar o salir de una ficha y, en ficheros de líneas, al entrar o salir de una línea.

3: Se ejecuta después de un alta o modificación tras decirle al básico *Correcto Sí*.

4: Proceso especial de impresión. Sólo puede haber uno por fichero. Si en un fichero hay un proceso de este tipo, en el mantenimiento standard se incluye la opción *Imprimir*, además de las ya conocidas *Alta*, *Consulta*, *Modificación*, *Baja*.

5: Es una fusión entre los tipos 2 y 3. La peculiaridad reside en la toma de datos de las líneas por parte de la cabecera, actualizándose con el alta de cada línea y, en caso de modificación, no será efectiva hasta su validación. Si al final del proceso tipo 5 le ponemos:

FSalida = "SINBARRA";

La barra de desplazamiento vertical de las líneas desaparece.

6: Funciona como el tipo 0, pero, a diferencia de éste, su ejecución es anterior a la comprobación de las relaciones entre ficheros, es decir, si estamos en el campo población (que está relacionado con la ficha de poblaciones), si el

proceso es de tipo 0, primero comprobará que exista el dato en la ficha de poblaciones y después ejecutará el proceso. Por el contrario, si es de tipo 6, primero ejecutará el proceso y después comprobará que exista el dato en la ficha de poblaciones.

7: Se ejecuta antes de entrar en el campo desde el cuál es llamado.

8: Se ejecuta al entrar en el ejecutable antes que nada. Es como si dijéramos un tipo 7 pero en lugar del campo, hace referencia al ejecutable completo.

9: Igual que el tipo 8, pero en lugar de ejecutarse al principio, se ejecuta al final.

10: Este proceso lo buscará el básico en caso de que no haya claves en el DEF o para casos en que no se quiera entrar en pantalla por cauces normales, por ejemplo para entrar datos desde un proceso. En este caso se entra en pantalla y se pone un [|ERROR](#); para que no salga la pantalla.

11: Se ejecuta en una ficha normal o de cabecera al salir con *CTRL+C* o al salir de la clave, y en una de líneas siempre que se sale del campo con el número de línea. Es particularmente útil cuando se hace un mantenimiento con [|ENTLINEAL](#).

13: En mantenimientos se ejecuta cada vez que se presenta una ficha por pantalla (cambio de registro), independientemente del tipo de operación que se esté realizando. Sólo puede haber uno por fichero.

14: Igual que el tipo 13, pero además se ejecuta al pasar por el campo donde está definido.

15: Grabación registro nuevo a bajo nivel.

16: Regrabación registro a bajo nivel.

17: Borrado registro nuevo a bajo nivel.

En los casos 15, 16 y 17 con |ERROR y FSalida a un numero positivo se devuelve ese error y no se efectúa la operación.

20: Es igual que el |TIPO 4; pero con opción a poder cambiar el literal, o sea, con el |TIPO 4 sale "Imprimir" y con el |TIPO 20 sale lo que asignes a FSalida.

Ejemplo:

|PROCESO Tipo20; |TIPO 20;

FSalida = "Hola Mundo";

|AVISO;

|FPRC;

30: Se ejecuta justo después de grabar la ficha al darla de alta o modificarla por el mantenimiento standard.

66: Este proceso se activa al pulsar F11 para consultar una relación.

80: de inicialización: Se ejecutan todos los tipos 80 que hayan antes de ningún mantenimiento o |programa (ojo el tipo 88 o 888 podría ejecutarse antes!!!).

90: de finalización: Se ejecutan todos los tipos 90 que hayan antes de cerrar la aplicación (liberar memoria etc).

88: se ejecuta al inicializar un def (al cargarse en memoria) se ejecuta uno por def que lo tenga declarado.

888: se ejecutan todos los 888 que haya cada vez que se inicializa un def tanto en el 888 como en el 88 Fichero@ va apuntando al def que se inicializa y FSalida tiene el path completo del def.

### **3.6.3.- Ficheros y campos.**

Como ya hemos indicado, la estructura de definición de ficheros en un programa o proceso hecho con el DSGen sigue la sintaxis *#nombrefichero #numero*.

Si queremos hacer referencia a este fichero desde un proceso, tenemos que indicarlo con una parrilla y, o bien el número, o bien el nombre. P.ej:

|[ABRE](#) #1;

|[ABRE](#) #ttma0001;

Si queremos hacer referencia al valor contenido en una variable, tendremos que indicarlo de tres modos posibles:

1: Haciendo referencia al número de fichero y al número de campo, ambos precedidos de una parrilla. P.Ej:

Si queremos hacer referencia al campo 3 del fichero que hemos definido como 2, escribiremos #2#3.

2: Haciendo referencia al nombre del fichero y al número de campo, ambos precedidos de una parrilla. P.Ej:

#ttma0001#2

3: Habiendo definido en el bloque |CAMPOS; su nombre, haremos referencia por este dato.

|CAMPOS;

#2#3 c\_direccion;

De este modo, siempre que escribamos *c\_direccion*, el básico interpretará que hacemos referencia al campo.

#### **3.6.4.- Variables.**

Toda la información intermedia, que sea necesaria para realizar los diferentes procesos, tendrá que ser almacenada en elementos auxiliares que son, precisamente las denominadas variables.

Disponemos de cinco tipos de variables:

-Numéricas.

-Alfanuméricas.

-Índice.

-Fecha.

-Puntero.

En la declaración, el nombre de la variable podrá tener como máximo 30 caracteres y la distinción entre los tipos de variables nombrados se produce en la inicialización de las mismas, la cual es obligatoria. En Diagram Software, S.L., en el departamento de producción existe una normativa a la hora de definir los nombres de las variables.

Las variables de tipo numérico comenzarán siempre por una letra *n* minúscula, seguida del nombre de la variable.

Las variables de tipo alfanumérico comenzarán siempre por una letra *a* minúscula, seguida del nombre de la variable.

Las variables de tipo fecha comenzarán siempre por una letra *f* minúscula, seguida del nombre de la variable.

Las variables de tipo puntero comenzarán siempre por una letra *p* minúscula, seguida del nombre de la variable.

Las variables de tipo externo comenzarán siempre por una letra *e* minúscula, seguida del nombre de la variable.

a) Para el caso de las variables numéricas se inicializarán bien a 0 o a cualquier número:

Por ejemplo:

`nRd = 5;`

`nX = 0;`

b) Las variables alfanuméricas lo harán bien a un string de caracteres o a un espacio en blanco, ambos delimitados por comillas. La longitud máxima que pueden tener es de 250 caracteres:

Por ejemplo:

`aNombre = "Juan Garcia";`

`aRespuesta = " ";`



En este caso, si se inicializa un string con un espacio en blanco no quiere decir que la máxima longitud que soporte sea de 1 carácter sino que puede, mediante concatenaciones, tener los 250 caracteres permitidos.

c) La inicialización de las variables fecha se producirá haciendo uso del símbolo ASCII "@" (Alt+64 en el teclado numérico)

Por ejemplo:

```
fFechaPago = @;
```

o incluso, asignándole un valor fecha por defecto:

Por ejemplo:

```
fFechaPago = @01.01.1990;
```

El formato fecha siempre es "dd.mm.aaaa" donde:

dd = dos dígitos para el día: 01,31,...

mm = dos dígitos para el mes: 01,12,...

aaaa = cuatro dígitos para el año: 1990 ...

d) Si a la declaración de la variable le precede un número entre llaves {}, indicamos que es una variable puntero. Si el número es mayor de cero, automáticamente, se crearán ese número de variables con el mismo nombre y tipo del puntero añadiéndoseles al nombre su número de orden. Es una manera de declarar masivamente variables iguales que podrán ser referenciadas por la variable puntero que las crea, se podrían considerar como arrays:

Por ejemplo:

```
{12}pMes = 0;
```

Se crearán 12 elementos de tipo numérico, inicializadas a cero y de componentes: mes1=0, mes2=0,..., mes12=0.

Si el número indicado es -1, sólo se creará el puntero que puede ser usado para referenciar cualquier variable. Un número cero equivale a no poner las llaves.

El número que contiene el puntero se puede asignar a una variable numérica normal cuyo nombre comienza con I más el nombre asignado al puntero, así podremos desplazarlos a lo largo de la variable puntero sin más que incrementar dicha variable numérica. La operación a realizar es `Ipuntero = variable<-;`

Con el ejemplo anterior :

```
{12}mes = 0; ||(Se crearan mes1 = 0,mes2=0, ...mes12 = 0;)
```

```
Imes = mes1<-;
```

```
PINTA 1020,mes; ||Es igual que pintar mes1
```

```
Imes = Imes + 1;
```

```
PINTA 1020,mes; ||Es igual que pintar mes2
```

#### **3.6.4.1.- Ámbito.**

Las variables, así como los ficheros, sólo son conocidas en un mismo fichero de procesos a menos que en su declaración se especifiquen como externas, para lo cual al nombre le precederá el símbolo "&" y será común a todos los procesos en los que ese mismo nombre sea declarado de igual forma, siendo también conocida en los informes. Es obligatorio que todas las variables estén definidas en este apartado.

Se inicia la zona de declaración de variables con la sentencia "`|VARIABLES;`" y se finaliza con la sentencia "`|FIN;`"

Las variables `nDeci_` se conservan y no se pierden por no estar declaradas en alguna ejecución.

Si una variable externa empieza por `PRMNT_` es permanente como las `nDeci_`

Su sintaxis general sería:

```
|VARIABLES;
```

```
nVarnum1= Numero;
```

```
aVaralfa1= "Texto";
```

fVarfecha = @;

{n}pVarpunt = *Numero* o *Texto*;

&eExterna1 = 0;

&PRMNT\_Persistente = Numero, Texto

o Fecha

|FIN;

Donde: "nVarnum" es nombre de variable numérica.

"aVaralfa" es nombre de variable alfanumérica.

"fVarfecha" es nombre de variable fecha.

"{j}pVarpunt" es nombre de variable puntero .

Ejemplo de una declaración de variables:

|VARIABLES;

fFecha = @;

aMensaje5 = "Factura : Procesando Albarán : Línea : Artículo :";

nPrimera = 0;

aMascara = "0000000";

aTempo = " ";

nLínea = 1;

nSolouno = 0;

{30}pDias = 0;

{12}pMensajes = " ";

{-1}pPunterogeneral = 0;

fFecha1 = @01.01.1990;

|FIN;

Dentro de un cálculo se pueden utilizar variables alfanuméricas y asignarles un valor precedido de una "A" o dentro de comillas.

Por ejemplo:

```
aNombre = APepe ;
```

```
aNombre = "Pepe Vazquez";
```

```
|PINTA aNombre;
```

*aNombre* tendría el siguiente valor

Pepe Vazquez

Si queremos usar las comillas dentro del texto podemos valernos de la variable

```
aComillas = A";
```

de la siguiente manera:

Por ejemplo:

```
aNombre = aComillas + "Pepe Vazquez" + aComillas;
```

```
|PINTA nombre;
```

Nombre tendría el valor

"Pepe Vazquez"

#### 3.6.4.2.- Variables reservadas.

A la hora de programar los procesos con el lenguaje DS, disponemos de una serie de variables reservadas que poseen ciertas propiedades útiles que a continuación detallamos:

- **Campo** : Numérica. En procesos de tipos 0, 6 y 7 contiene el número de campo desde donde se llama al proceso.
- **Fichero@**:

Nueva función de DSGen: septiembre de 2000 → básico  
9.9 W-1

Existe una nueva variable reservada e interna llamada **Fichero@** , el funcionamiento es el mismo que la que siempre ha existido, **Campo** , pero en vez de llevar el número de campo lleva el def correspondiente a ese campo.

Ejemplo:

este proceso lo podremos llamar desde cualquier campo que sea alfanumérico y que queramos rellenar de ceros a la izquierda, lo declararíamos en un .cal especial y solamente tendríamos que incluirlo en la lista de .cal del def y en la llamada del campo (ERellenaCliente).

```
|VARIABLES;  
aAlfa1 = "";  
  
|FIN;  
  
|PROCESO RellenaCliente; |TIPO 0;  
aAlfa1 = #Fichero@#Campo;  
  
|QBF aAlfa1;  
aAlfa1 = ("000000" + aAlfa1) % -106;  
#Fichero@#Campo = aAlfa1;  
  
|PINTA #Fichero@#Campo;  
  
|FPRC;
```

- **Contenido:** Alfanumérica. En procesos de tipos 0 y 6 contiene el valor del campo desde donde se llama al proceso, antes de ejecutarlo.

- **Control:** Alfanumérica. En las posiciones del 1 al 9 contiene el nombre del fichero cuya pantalla esta pintada en ese momento. En la posición 10 el número de pantalla, (puede haber varias). En las posiciones 20, 21 hay un SI o un NO que indica si queremos utilizar los flags de visualización, modificación, etc... del campo (véase capítulo AGIGEN ). En las posiciones 22, 23

hay un SI o un NO que indica si queremos repintar las líneas (de un fichero de líneas) por alguna razón.

- **FEntrada** : Es un flag de entrada. Útil para procesos llamados desde mantenimientos indicándonos el modo de los mismos ( si alta, modificación...). El modo correspondiente lo conoceremos, no directamente del valor de FEntrada, sino a través de dos operaciones. El ámbito de la primera será general, es decir aplicables a todos los campos del registro y viene dada por la siguiente expresión:

$$nN = (FEntrada / 100) ! 0)$$

donde: nN es el valor resultado de dividir la variable reservada FEntrada por 100 y redondeado a cero decimales.

La segunda operación se considera de ámbito local pues indicará el modo en que ha entrado el campo. Viene representada por la siguiente expresión:

$$nM = FEntrada \$ 100$$

donde: nM corresponde al valor de FEntrada módulo 100 (valor de los dos últimos dígitos).

Por ejemplo, al dar de alta una ficha los campos se introducen en modo alta siempre que sea la primera vez, si retornamos antes de validar al campo en cuestión para rectificarlo, éste pasa a encontrarse en modo modificación.

Los valores que tomarán tanto nN como mM serán:

1 – Alta.

2 - Modificación (Nota: las líneas se tratan como fichas).

3 – Baja.

4 – Consulta.

77 - Diferido (Procesos de relaciones, defectos).

- **FSalida**: El valor que toma esta variable dependerá de la función que haya sido invocada. Por lo general, *FSalida* controla los posibles errores de ejecución de la mayor parte de funciones de la librería. Los valores que puede tomar son los siguientes:

FSalida = 0 ,es que todo va bien.

FSalida = -1, por algún motivo no se ha ejecutado correctamente la instrucción.

FSalida < 100, si se han detectado errores relativos al sistema operativo.

FSalida > 99, si se han detectado errores relativos a ficheros indexados.

Cuando tratamos con funciones de entradas de datos o al ejecutar la función [PAUSA](#) los valores devueltos por la variable FSalida son los siguientes:

0 = Return

1 = Escape

2 = Flecha Arriba

3 = Flecha Abajo

4 = Página Arriba

5 = Página Abajo

6 = Consulta

7 = Abortar (CTRL-C)

8 = Ayuda (HELP)

9 = Función 1

10 = Función 2

11 a n = Función 3 a n (hasta 18 funciones)

- ***E<sub>sup</sub>*** : Alfanumérica. donde se indica el valor superior al estilo de la entrada de campos en los def, de la operación de entrada de datos directa "?".

- ***E<sub>inf</sub>*** : Alfanumérica. donde se indica el valor inferior al estilo de la entrada de campos en los def, de la operación de entrada de datos directa "?".

- ***Pos*** : Variable numérica polivalente. Su uso es muy variado y su valor dependerá de la situación en la que se encuentre. En las funciones de ficheros directos se usa para posicionar el byte del fichero a partir del cual leer o grabar. Para las funciones de entrada de campos o pintar campos indica o modifica la posición original del campo.

- ***Usuario.***

Contiene el nombre del usuario que está utilizando los programas.

- ***Impresora.***

Esta variable contiene el nombre de la impresora que se ha asignado con la instrucción [|IMPRE](#). Se puede asignar a esta variable un nombre de impresora y después con la instrucción [|IMPRE](#) -1 no nos pide la configuración de la impresora.

- ***Parámetro***

Contiene los parámetros al invocar un programa llamado desde el menú o desde la instrucción [|CORRE](#). Si el proceso se efectuara desde la instrucción [|ALARMA](#) entonces contiene la fecha y hora de activación.

Hay una serie de variables usadas de intermediarias en las operaciones, que van de la 8 a la 30 para operaciones numéricas, y de la 31 a la 53 para operaciones alfanuméricas, estas solo se pueden usar con un puntero y con cuidado pues podrían afectar a los resultados de las operaciones si estas contienen varios operadores por formula.



Las variables definidas por el usuario van a partir de la 54 en adelante según el orden en que se van declarando.

#### **3.6.4.3.- Variables persistentes.**

Si en lugar de declarar una variable anteponiéndole el símbolo '&', le antepone además la palabra 'PRMNT\_', esta variable se convierte en persistente y se podrá disponer de ella en cualquier aplicación. Se puede declarar en cualquier parte, surtirá efecto a partir de abrir el primer .tab que la contenga declarada y se mantendrá hasta que se salga del básico.

Es especialmente útil para mantener información presente siempre o intercambiar información entre aplicaciones, ya que se mantendrá aunque se cambie de aplicación o se haga un |SUB\_EJECUTA de un .tab de otra aplicación (p.ej. desde dscomer9 hacer un |SUB\_EJECTUTA ':contagen/dsapunte.tab').

P.ej.

```
|VARIABLES;  
  
    &PRMNT_aPersistente = "Hola";  
  
    &PRMNT_nPersistente = 123;  
  
    &PRMNT_fPersistente = @;  
  
|FIN;
```

#### **3.6.5.- Operaciones.**

Las operaciones se definen con un campo o variable al cual se le asigna otra variable o campo o el resultado de una serie de operaciones con campos y/o variables. Las operaciones se pueden realizar en modo numérico, alfanumérico o fecha. El modo viene determinado por la naturaleza del resultado, si el resultado es una variable/campo numérica la operación se hará

en modo numérico, habiendo una conversión automática de todas las variables/campos que no coincidan con este modo. Si el resultado es alfanumérico pasará lo mismo pero con las fechas hay un matiz, pues la conversión de fecha a numérico da el total de días que tiene la fecha desde el año 0, y viceversa, la conversión de un numérico a fecha da la fecha que correspondería considerando el número como un total de días. SOBRE TODO las operaciones fecha sólo realizan la conversión a tipo fecha de los operandos CUANDO LA OPERACIÓN SEA SIMPLE del tipo  $f = f1 + n$ ; de haber más, los operandos se convierten a tipo alfanumérico excepto en la última operación a realizar, que se pasa a fecha y se asigna al resultado tipo fecha. Como no es fácil determinar cuál es la última operación, se recomienda el uso de los paréntesis.

Por ejemplo:

$f = \text{variable que pasará a fecha} + (\text{alfa1} + \text{numero1} \% \text{numero2})$  que se hará en alfanumérico y cuyo resultado pasará a fecha; así que en modo fecha solo operaría el primer '+'.  
El operador PARÉNTESIS puede modificar el modo de las operaciones incluidas entre ellos con la siguiente sintaxis:

$a [\text{numérica}] = b [\text{numérica}] + [\text{operación\_numérica}] (A \setminus c + d) [\text{operaciones alfanuméricas}]$

$a [\text{alfanumérica}] = b [\text{alfanumérica}] + [\text{operación alfanumérica}] (N \setminus c + d) [\text{operaciones numéricas}]$

$(A \setminus \text{operaciones} \dots)$  -> entre paréntesis operaciones alfanuméricas.

$(N \setminus \text{operaciones} \dots)$  -> entre paréntesis operaciones numéricas.

Los operadores que utilizaremos en las diferentes operaciones se clasificarán en:

Operador de asignación: "=" igual

Operador según prioridad:

1 "()"paréntesis

2 "?" interrogación

"<"apuntador

"~" tilde

"&" ampersand

3 "\*" asterisco

"/" barra

"%" porcentaje

"<" menor que

">" mayor que

!" admiración

"\$" dólar

4 "+" positivo

"-" negativo

(Operador modificador de operandos): "."utilizado básicamente en procesos de tipo 2.

#### **3.6.5.1.- Numéricas.**

Esta es la relación de operadores matemáticos disponibles en los procesos:

a + suma.

b) - resta.

c) \* producto.

d) / división.

e) % porcentaje.

f) > suma porcentaje.

g) < resta porcentaje.

h) ! redondeo.

i) \$ resto módulo.

j) = asignación.

k) ? entrada directa desde teclado. (numérica)

l) <- obtener el número interno de la variable para usarla desde un puntero.

m) & valor ASCII de un carácter o carácter correspondiente a un valor ASCII.

n) . Convierte en el caso de proceso en modo reversible (ver tipos de procesos) el + en - el - en + el \* en / el / en \*.

o) ~ Pone cero en el resultado.

Mostraremos algunos ejemplos de los operadores menos evidentes y conocidos:

a = 100 % 10; calcula el 10% de 100

a = 40 > 10;                      a = 40 + 40%10;

a = 40 < 10;                      a = 40 - 40%10;

a = 10.163 ! 1;                      a=10.2

a = 10 \$ 3;                      a=1

### **3.6.5.2.- Alfanuméricas.**

Las operaciones alfanuméricas que normalmente suelen realizarse en los procesos son las siguientes:

a) + concatenación. (concatena dos cadenas)

b) - sustracción. (elimina una subcadena)

c) \* repetición.

d) / 'corte' calculado.

e) % multiuso.

f) > paso a mayúsculas.

g) < paso a minúsculas.

h) ! print con formato.

i) \$ resto módulo.

j) = asignación.

- k) ? entrada directa desde teclado. (alfanumérica)
- l) & carácter ASCII correspondiente al número que sucede al operador.
- m) . Convierte en el caso de proceso en modo reversible (ver tipos de cálculos) el + en -, el - en +, el \* en /, el / en \*.
- n) ~ trunca el resultado a variable vacía.

Para poder entender mejor ciertas operaciones, no comunes, mostraremos junto a sus descripciones algunos ejemplos:

- a) Si nom1 = "Juan " y nom2 = "Garcia"  
nom3 = nom1 + nom2 = "Juan Garcia".
- b) Si nom1 = "Parkinson" y nom2 = "rkin"  
nom3 = nom1 + nom2 = "Pason" .
- c) var3 = var1 \* n ;

El primer operando es alfanumérico y el segundo numérico, repite un numero de veces una cadena:

- Si nom2 = "Pason ", nom3 = nom2 \* 4, vale  
"Pason Pason Pason Pason ".
- d) var3 = var1 / n ;

El primer operando es alfanumérico y el segundo numérico. Divide la longitud del primer operando por el segundo y del número obtenido extrae la parte entera. Luego toma este número de caracteres empezando por la izquierda, del primer operando.

Por ejemplo:

Si nom2 = "Empresario" , nom3 = nom2 / 3 vale "Emp".

- e) var3 = var1 % An ;

El primer operando es alfanumérico y el segundo numérico.

Casos:

( n = 0 ) Devuelve el número de caracteres de var1.

( $0 < n < 100$ ) Devuelve la cadena que queda a partir de la posición  $n$ .

( $n > 100$ ) Devuelve un número de caracteres a partir de una posición.

Ejemplo:

$n = 503$  devuelve 3 caracteres a partir de la posición 5.

$n = 101$  devuelve el primer carácter.

( $n < -100$ ) Igual que para  $n > 100$  empezando a contar por el final.

Ejemplo:

`var1 = "Miguel queria un Golf"`

`var3 = var1 % A606` vale "quería".

`var3 = var1 % -A104` vale "Golf".

f) `var3 = var1 >`; Un sólo operando.

`var1 = "domingo".`

`var3 = var1 >` vale "DOMINGO".

g) `var3 = var1 <`; Un sólo operando.

`var1 = "DOMINGO".`

`var3 = var1 <`; vale "domingo".

h) `var3 = var1 ! var2;`

Equivale a la función `sprintf(var3,var1,var2)` del C, que permite asignar a la variable buffer `var3`, el contenido de la variable `var2` bajo el formato "%ns", es decir alfanumérica y con un corte de  $n$  caracteres.

### **3.6.5.3.- Con fechas.**

Como ya sabemos, una variable puede ser declarada como fecha cuando se le asigna el símbolo @. También, puede asignarse un campo que contenga una fecha a una variable numérica cuyo valor equivale al número de días desde la fecha 00.00.0000. Luego se puede asignar este número a una variable fecha, obteniendo de nuevo una fecha o sumar este número a una variable fecha ...

O sea, que la flexibilidad a la hora de trabajar con este tipo de variables es total. Las operaciones que disponemos cuando programemos procesos son las siguientes:

- a) + Sumar días, meses, años a una fecha.
- b) - Restar fechas o restar días, meses, años a fechas.
- c) ~ Asignar la fecha del sistema a una variable fecha.
- d) % Devuelve la fecha escrita en letras.

Describiremos con ejemplos las operaciones mencionadas:

a) + Para sumar a una fecha un número D de días, un número M de meses y un número A de años, realizaremos la operación:

$$j = D + (N \setminus M / 1000) + (N \setminus A / 100000)$$

Si el resultado se asigna a una variable fecha el valor obtenido adquirirá su tipo.

Por ejemplo:

fecha3 = fecha1 + 1.003002; Suma a fecha1 2 años, 3 meses y un día.

fecha3 = fecha2 + fecha1; Da una fecha correspondiente a la suma de los días de las fechas operandos, a partir de la fecha 00.00.0000. y convertida al tipo de variable resultante.

días = fecha1 + fecha2; Daría la suma de días ... por ser días una variable numérica.

b) - Se puede restar a una fecha un número de días, meses y años, con el formato descrito, para obtener el número de días de diferencia, asignando el resultado a una variable numérica.

c) ~ Asigna la fecha del sistema a una variable fecha.

Sintaxis : fecha = ~;

d) % Pasar a letras fechas:

Sintaxis : fecha = fecha1 % t; donde t puede valer:

0 - Entrega el día de la semana.

1 - Entrega el mes.

2 - Entrega la fecha completa.

Por ejemplo: si fecha1 = "01.01.1989"

fecha = fecha1 % 0;

Obtenemos fecha = "01.01.2000LUNES"

Otro ejemplo: fecha = fecha1 % 2;

Cuyo resultado será:

fecha = "01.01.2000Lunes 1 de Enero de 2000"

### **3.6.6.- Control de Flujo de programas.**

Normalmente, la ejecución de un proceso se realiza secuencialmente, es decir una instrucción se ejecuta después de otra. Ahora bien, en determinadas circunstancias ese flujo secuencial puede modificarse y ser desviado en otra dirección bajo ciertas condiciones. El control de flujo de un proceso puede ser modificado por:

- Bifurcaciones condicionales.
- Bifurcaciones incondicionales.
- Bucles de instrucciones.
- Bucles de lectura de ficheros.
- Anidación.

#### **3.6.6.1.- Bifurcaciones condicionales.**

Se dice que se ha producido una bifurcación condicional en el flujo de un proceso cuando, bajo determinadas condiciones, ese flujo cambia de dirección.

Disponemos de una serie de operadores para modificar el curso de un proceso, que se clasifican en:

- \* Operadores condicionales.
- \* Operadores lógicos.



### **3.6.6.1.1 LOS OPERADORES CONDICIONALES.**

Estos operadores permiten realizar comparaciones de valores y dependiendo de que se cumpla la condición dada o no, ejecutarán una serie de instrucciones u otras.

Los operadores condicionales son los siguientes:

- 1) = Igual a.
- 2) ! Distinto de .
- 3) < Menor que.
- 4) > Mayor que.
- 5) ] Mayor o igual que.
- 6) [ Menor o igual que.

La sintaxis a seguir es : <var1> op <var2>;

donde: var1 y var2 son los operandos. En estos casos no se permite la utilización de paréntesis.

### **3.6.6.1.2 LOS OPERADORES LÓGICOS.**

Los operadores lógicos permiten anexar un número de condiciones y dependiendo del enlace entre ellas se ejecutarán o no las instrucciones que les siguen.

Estos operadores lógicos son:

|Y; Operador AND.

|O; Operador OR.

La sintaxis que exigen dichos operadores es la que se muestra:

Condición1;|Y Condición2; .....;|Y CondiciónN;

Condición1;|O Condición2; .....;|O CondiciónN;

En el primer caso, la bifurcación se llevará a cabo si y sólo si se cumplen las N condiciones, es decir desde condición1 hasta condiciónN, si alguna de ellas es falsa o no se cumple la bifurcación no se llevará a cabo.

En el segundo caso, la desviación se producirá siempre que alguna de las condiciones se cumpla, pero no necesariamente todas sino como mínimo una y como máximo las N presentes.

Una vez conocidos los operadores condicionales y lógicos utilizados en este tipo de bifurcaciones, pasamos a mostrar la instrucción característica de las desviaciones condicionales y que se basa, principalmente, en los operadores descritos.

La sintaxis de una bifurcación condicional es:

|SI operando1 <operador cond.> operando2; (instrucciones)

|SINO;

(instrucciones)

|FINSI;

donde: operador cond. es el operador condicional.

Si hacemos uso de operadores lógicos, esto podrán ser mixtos, es decir que podremos anexar condiciones que operen con el operador |Y con otras que lo hagan con el operador |O.

En este caso las condiciones se verificarán de izquierda a derecha y de dos en dos de tal manera que si comenzamos con un |Y y la primera condición no se cumple, ya no se verifica nada más dándose por falsa la condición. Por otra parte, en caso de cumplirse se verifica la siguiente condición que, aunque no se cumpla, siempre que el próximo operador lógico sea |O el resultado se considerará como válido.

Para verlo más claro, consideremos el siguiente ejemplo:

|SI 1 = 1;|Y 1 = 2;|O 2 = 2; -> Se cumple

|SI 1 = 2;|Y 1 = 1;|O 2 = 2; -> No se cumple

|SI 1 = 1;|Y 1 = 2;|O 2 = 3; -> No se cumple

Recalcamos de nuevo que, los enlaces entre diferentes condiciones a través de operadores lógicos se realizan SIN UTILIZAR PARÉNTESIS.

Si la condición se cumple, se ejecutan las instrucciones que siguen al "|SI" hasta encontrar un "[|FINSI](#);" que indica final del ámbito de la condición. Si no se cumple la condición inicial, la ejecución del programa continúa en la primera instrucción después del "[|FINSI](#)". Podemos incluir una instrucción intermedia entre estas dos mencionadas denominada "|SINO;" que, en caso de no cumplirse la condición, permitirá la ejecución de las instrucciones que le siguen y continuará el programa normalmente. Se consideraría como una "bifurcación doble" en la que, antes de proseguir con la ejecución del programa, actuará de una forma u otra frente a la validación de la condición.

Cabe la posibilidad de anidar varias instrucciones de bifurcación, por ejemplo:

```
|SI condición;  
    ( instrucciones )  
|SINO;  
    |SI condición;  
        (instrucciones)  
    |SINO;  
        (instrucciones)  
|FINSI;  
    (instrucciones)  
|FINSI;
```

#### **3.6.6.2.- Bifurcaciones incondicionales.**

Hablaremos de bifurcaciones incondicionales cuando el flujo del proceso sea desviado sin ningún tipo de restricción o condición. Dicha bifurcación se puede realizar bien dentro de un mismo proceso o entre diferentes y dentro de un mismo fichero.

Las instrucciones básicas para ejecutar estos cambios en la dirección de un proceso son las siguientes:

[|ET](#) nometiqueta; señala un punto del proceso con el nombre 'nometiqueta', etiqueta ese punto, para volver a él en un momento determinado.

[|VETE](#) nometiqueta; hace que la siguiente instrucción a ejecutar sea la siguiente al punto etiquetado (con [|ET](#)) con el nombre indicado en 'nometiqueta'.

### **3.6.6.3.- Bucles de instrucciones.**

En un momento dado y a lo largo del proceso, puede interesarnos realizar un cálculo un número determinado de veces, por ejemplo realizar un acumulado de una variable. Para ello disponemos de una serie de instrucciones que poseen la siguiente sintaxis:

[|PARA](#) [operación]; [|SI](#) [condición]; [|HACIENDO](#) [operación];

( instrucciones )

[|SIGUE](#);

Tanto las operaciones como las condiciones son opcionales, se pueden omitir, pero las instrucciones nunca.

Veamos cómo se ejecuta este bucle de instrucciones.

Primero se ejecuta la operación del "[|PARA](#)", verificándose después la condición del "[|SI](#)". Si ésta se cumple, se pasa a ejecutar la siguiente instrucción después del "[|HACIENDO](#)" (no nos referimos a la operación del [HACIENDO](#) ) y las sucesivas hasta encontrar la instrucción: "[|SIGUE](#);" que marca el final del bucle, donde se ejecuta la operación del "[|HACIENDO](#)" verificando después la condición del "[|SI](#)" y ,si ésta se cumple, se sigue la ejecución en la primera instrucción después del [HACIENDO](#). Si la condición no se cumple se continúa la ejecución del proceso en la primera instrucción después del "[|SIGUE](#)".

Por ejemplo:

|PARA i = 0; |Si i < topei; |HACIENDO i = i + 1;

( instrucciones.....)

|SIGUE;

Si quisiéramos interrumpir por cualquier motivo la ejecución del bucle, sólo tendríamos que incluir una orden |SAL;

#### **3.6.6.4.- Bucles de ficheros.**

A continuación presentamos, en los dos próximos apartados, dos tipos de bucles para la lectura de forma secuencial de ficheros. El primero es el denominado bucle declarado y el segundo el de lectura de líneas.

##### **3.6.6.4.1.- Bucle declarado.**

De forma análoga a la mostrada para declarar ficheros, campos o variables, podemos declarar bucles de procesado de ficheros.

En este bucle podremos indicar el orden en que deben leerse los registros y marcar unos límites de tal manera que sólo se lean ciertas fichas cuyos valores nos interesan y no todas. Para cada ficha que cumpla las condiciones definidas se ejecutarán una serie de cálculos.

La sintaxis es la siguiente:

|DEFBUCLE;

#n#c;

c1,c2,c3,...,cn;

#f1#c1,#f1#c2,...,#f1#cn;

#f1#c10,f1#c11,...,#f1#c1n;

be;

cal1,cal2,cal3,cal4,cal5,cal6,cal7,cal8;

|FIN;

donde: #n#c; es el grupo fichero-clave: n es el número de fichero donde se desea leer y c es la clave que utilizaremos para leerlo. En caso de ésta

última tomar el valor 0, nos indicará que la lectura la haremos a partir de los campos declarados en la siguiente línea.

c1,c2,c3,...,cn; En caso de querer tener en cuenta campos diferentes a los de la clave, los indicaremos en esta línea. Si con la clave es suficiente los omitiremos escribiendo sólo un ;. Ahora bien aunque el valor de la clave sea 0 podremos omitirlos en cuyo caso la lectura será de forma desordenada.

#f1#c1, #f1#c2, ..., #f1#cn; A la hora de realizar una lectura secuencial de un fichero, es necesario establecer unos límites para los valores que conformarán su ordenación. En esta línea definiremos el límite inferior tanto de la clave a utilizar como de los campos seleccionados diferentes a la primera. Aunque el formato indicado corresponde a la representación de un campo, también se admiten variables que podrían ser resultados de operaciones con los campos, por ejemplo.

#f1#c10, #f1#c11,..., #f1#c1n; Es evidente que, si definimos unos valores mínimos, deberemos también especificar unos valores máximos con tal de delimitar la ordenación. La inclusión de estos valores es obligatoria y si el algún caso se deseara leer el fichero a partir de un valor mínimo únicamente, se escribiría el valor más alto posible como máximo.

be; En un momento determinado podemos querer bloquear cada una de las fichas seleccionadas con el fin de restringir su acceso a otros usuarios mientras las manipulamos. Ello lo indicaremos con la letra b. Ahora bien, en el momento de lectura puede que algún registro haya sido bloqueado con anterioridad en una operación similar o en otra de lectura y/o escritura. En ese caso tenemos dos opciones, o bien ignoramos el o los registros bloqueados o bien procedemos a esperar, si el registro en cuestión nos interesa, hasta que sea desbloqueado. Esta última opción la activaremos con la letra e.

pro1, pro2, pro3, pro4, pro5, pro6, pro7, pro8; En esta línea especificaremos los procesos que ejecutaremos con los registros que vayamos leyendo. Todos los procesos deberán haberse definido previamente. La

función que tendrán cada uno de los procesos dependerá del orden que ocupen en la línea. Dicho orden lo describimos a continuación:

pro1.- Este proceso se ejecutará antes de empezar la lectura del fichero, se puede utilizar para inicializar las variables o campos que marcan los límites de la lectura. Si durante su ejecución se produce un error, invocación a la función [|ERROR;](#), se impedirá la denominada doble ordenación.

pro2,pro3,pro4.- Estos tres procesos se ejecutarán para cada registro leído válido, siempre que se encuentre entre los límites marcados.

Si por cualquier motivo se produce un error durante el desarrollo del proceso pro2 ( [|ERROR;](#) ) se saltará al siguiente registro. Si existe pro3 y el fichero posee enlace con su homólogo de líneas, se ejecutará para cada línea una vez. En caso de error también saltará al siguiente registro. La ejecución de pro4 se producirá después de los dos anteriores y siempre que en ninguno de ellos se haya dado un error.

pro5.- La ejecución de este proceso se llevará a cabo tras la exploración de todo el fichero, como proceso final.

pro6, pro7, pro8.- Estos tres procesos dependerán directamente del resultado de la ejecución de pro1, si éste da error no se ejecutarán. Por otra parte, el ámbito de éstos es el mismo que el de los procesos pro2, pro3, pro4 con la excepción de ejecutarse sobre la denominado doble ordenación.

Todos aquellos procesos que no necesitemos para la lectura de un fichero podremos omitirlos indicando NULL en vez del nombre del proceso siempre que su posición sea intermedia. Si es final se puede trincar la línea con un ";". Por ejemplo,

Pos. intermedia pro1,pro2,NULL,pro4,pro5,pro6,pro7,pro8;

Pos. final pro1,pro2,pro3,pro4;

En el primer caso hemos omitido el proceso número 3 y en el segundo, los últimos cuatro procesos.

Por ejemplo:

Dado un fichero que se llama *ttma0001* que corresponde a clientes y tiene los campos:

0	Codigo	A	6
1	Nombre	A	40
2	Telefono	A	15
3	Nif	A	15

Queremos hacer un bucle que nos rellene un fichero temporal que se llama *temp0001* y que tiene los campos:

0	Nombre	A	40
1	Nif	A	15

Pero sólo de los clientes que estén entre los márgenes que determine el usuario en nuestro DEF actual que se llama *ttma0002* y tiene los siguientes campos:

0	DesdeCodigo	A	6
1	HastaCodigo	A	6
2	DesdeNombre	A	40
3	HastaNombre	A	40
4	DesdeTelefono	A	15
5	HastaTelefono	A	15

Crearemos un Def que contenga estos cinco campos. En procesos haremos el siguiente *programa*:

|FICHEROS;

ttma0002 #0;

ttma0001 #1;

temp0001 #2;

|FIN;

|PROCESO ttma0001;

#2#0=#1#1;



|[LEE](#) 000#2=;

|[SI](#) FSalida!0;

    #2#1=#1#4;

    |[GRABA](#) 020#2n;

|[FINSI](#);

|FPRC;

|DEFBUCLE ttma0001;

    #1#1;

    1,2;

    #0#0,#0#2,#0#4;

    #0#1,#0#3,#0#5;

    be;

    NULL,ttma0001;

|FIN;

|PROGRAMA;

    |[CLS](#);

    |[CABEZA](#) "PROGRAMA";

    |[PINPA](#) #0#0;

    |[ABRE](#) #0;

    |[DELINDEX](#) #0;

    |[CIERRA](#) #0;

    |[ABRE](#) #0;

    |[ENDATOS](#) #1#0;

    |[SI](#) FSalida=0;

        |[ABRE](#) #temp0001;

        |[BUCLE](#) ttma0001;

|[CIERRA](#) #temp0001;

|[FINSI](#);

|[CIERRA](#) #0;

|FPRO;

Notar que el bucle está definido antes que el programa, que el proceso que llama el bucle se llama igual que éste y que está definido antes del bucle.

## **LA DOBLE ORDENACIÓN.**

Si con la ordenación de los campos del fichero indicado en la primera línea no tenemos suficiente, podemos incluir una línea más a la declaración del |DEFBUCLE en la cual indicaremos, separados por comas, los campos (de cualquier fichero) o variables cuyos valores iremos asignando, durante la ordenación normal, a través de los procesos pro2, pro3, pro4 y por los que queremos tener una lectura ordenada del fichero indicado en la primera línea (el principal).

Lo veremos más claro con un ejemplo. Supongamos que disponemos de un fichero donde están almacenados todos los clientes de nuestra empresa. Cada cliente posee una serie de campos significativos, entre ellos su código, nombre, calle,... y por ejemplo, el número de representante que lo atendió. Por otra parte, deberemos disponer de un fichero donde se hallen todos los representantes de nuestra empresa con su número y nombre, relacionado con el de clientes. Nuestro objetivo es ordenar todo el fichero de clientes, no por el número de representante sino por el nombre del mismo pues nos interesa una lista de clientes atendidos por cada uno de nuestros representantes. Como podemos observar estamos tratando con campos de otros ficheros y por lo tanto deberemos realizar una doble ordenación. Para ello, la ordenación inicial será en base al código del cliente y después, en el proceso pro2 se leerá el nombre de representante y se volverá a ordenar el fichero ejecutándose los procesos pro6, pro7 y pro8 en vez de pro2, pro3 y pro4.

Para realizar la doble ordenación escribiremos una línea con la siguiente sintaxis:

c1, c2, c3, c4, c5,...;

Esta línea se puede incluir o no siendo la única opcional en la declaración y cuya inclusión determina una segunda lectura del fichero ordenada según los campos aquí indicados y ejecutando los procesos pro6, pro7, pro8 tras la cual se ejecutará el proceso pro5 otra vez.

Si a lo largo del proceso deseamos abortar la lectura de un DEFBUCLE incluiremos en uno cualquiera de los procesos pro2, pro3, pro4, pro6, pro7 o pro8 la función "[|ERROR10](#);" que provocará el abandono de la lectura justo después de finalizar el proceso que la ejecuta.

Hasta ahora habíamos descrito el funcionamiento de la estructura de la función "[|DEFBUCLE](#);" pero para poder ejecutar el bucle hay que hacer uso de la función "[|BUCLE n](#);" donde n es un número de orden a partir de cero según se hayan declarado los bucles "[|DEFBUCLE](#);", es decir si hemos definido cuatro bucles de lectura de ficheros el valor n estará acotado entre 0 y 3 y procesaremos cada uno en el momento que nos interese llamando a la función anterior.

#### **3.6.6.4.2.- Bucle de lectura de ficheros de líneas.**

En el apartado anterior describimos una forma de lectura de ficheros en la que se ordenaba al tiempo de leerse. A continuación, presentamos otro tipo de bucle de lectura pero en este caso indicado, exclusivamente, para ficheros de líneas con la única ordenación fija según el número de línea.

La función que utilizaremos posee la siguiente sintaxis:

[|BUCLELIN](#) mnombreproceso#n;

donde: n es el número de fichero.

Esta función nos permite leer las líneas (por orden de número de línea) de un registro de tipo "cabecera" que previamente habremos leído, ejecutándose el proceso "nombreproceso" para cada línea que es leída según el modo m:

m = 0 lee bloqueando.

m = 1 lee bloqueando borra las líneas si el proceso no tuviera ninguna operación, únicamente borraría

m = 2 lee sin bloquear.

Con "[|ERROR;](#)" se interrumpe la lectura de las líneas.

#### **3.6.6.5.- Anidación de procesos.**

El lenguaje DS también permite crear anidaciones de diferentes procesos. De esta forma podremos ejecutar, bajo ciertas condiciones, cualquier proceso desde el actual y tras su ejecución seguir con las instrucciones posteriores a la llamada.

La función que nos permite consumir las anidaciones es la siguiente:

[|HAZ](#) nombreproceso;

donde: 'nombreproceso' es el nombre del proceso a ejecutar desde el actual. Este proceso deberá estar contenido en el fichero ejecutable aunque pertenezca a otro fichero de procesos.

### **3.7.- Rutinas.**

Las diferencias entre las rutinas y los procesos son:

Las rutinas no pueden llamarse desde un DEF con la orden Exxxxx.

Las rutinas no tienen un tipo de evento.

Por lo demás se con un [|HAZ](#) y su sintaxis es la siguiente:

[|RUTINA](#) rCalculo;

... || Aquí pondremos las órdenes de la rutina

[|FRUT;](#)

#### 4.- EL EDITOR DE TEXTOS DEL GENERADOR.

El GENERADOR dispone de un editor de textos integrado.

Los comandos utilizados en la edición de textos son los siguientes:

<b><u>Comando</u></b>	<b><u>Descripción</u></b>
F1 o CTRL-Y	Borra la línea corriente.
F2	Recupera la última línea o bloque borrado.
F3 o CTRL-Q R	Sitúa al principio del texto.
F4 o CTRL-Q C	Sitúa al final del texto.
F5 o CTRL-Q F	Busca un texto, una vez encontrado lo ilumina y sise pulsa 'B' sigue buscando el patrón, si no finaliza la búsqueda.
F6 o CTRL-Q A	Busca un texto y lo reemplaza. Cuando encuentra el texto se puede pulsar 'B' para seguir buscando, 'R' para reemplazar el corriente y seguir buscando, 'T' para reemplazar todos los que se encuentren u otra tecla para abandonar.
F7 o CTRL-K B	Marca el inicio del bloque.
F8 o CTRL-K K	Marca el final del bloque.
F9 o CTRL-O L	Entra márgenes en el texto.
F10	Imprime el texto.
CTRL-O J	Justifica el texto.
CTRL-Q B	Va al principio del bloque.
CTRL-Q K	Va al final del bloque.
CTRL-K C	Copia el bloque marcado en la posición del cursor actual.
CTRL-K V	Mueve el bloque a la posición del cursor actual.
CTRL-K Y	Borra el bloque marcado.

CTRL-K R	Lee un fichero y lo marca como bloque.
CTRL-K W	Graba el bloque marcado en disco debiendo asignarle un nombre al fichero.
Insert	Activa o desactiva el modo de inserción.
Del	Borra el carácter en la posición del cursor desplazando el resto de la línea hacia la izquierda. En una línea en blanco borrará ésta.
Backspace (< -)	Borra hacia atrás.
Return	Salta a la siguiente línea. En modo de inserción añadirá una línea.
PgUp	Avanza una página del texto.
PgDn	Retrocede una página del texto.
Esc	Sale de la edición del texto validando su contenido, es decir, grabándolo en disco.
CTRL-C	Abandona, sin salvar, la edición. No valida las posibles modificaciones producidas.

## 5.- GUIA DE REFERENCIA.

En esta Guía de Referencia de Sentencias del lenguaje DS se describirán, por orden alfabético, cada una de las órdenes integradas en el entorno de programación.

La estructura que tomará cada una será la siguiente:

<NOMBRE DE LA SENTENCIA>

SINTAXIS: |<NOMBRE DE LA SENTENCIA> [parámetros];

DONDE: (Explicación de los parámetros si los hubiere) ...

PROPÓSITO: Explicación del propósito de la sentencia.

DEVUELVE: Descripción de los posibles valores devueltos por la sentencia.

VER: referencias a otras sentencias relacionadas con ésta.

NOTA.

*Todas las sentencias descritas en esta guía de referencia deben escribirse en mayúsculas a la hora de incorporarlas en un proceso y comenzarán por el símbolo ASCII '|', número 124 , llamado "pipe" o "cauce".*

## ABORTA

SINTAXIS: |ABORTA;

PROPÓSITO: El objetivo de la sentencia |ABORTA es finalizar la aplicación en el momento en que se ejecuta. Su ámbito se restringe a los procesos de tipo PROGRAMA. Si se ejecuta desde un proceso anidado sólo será efectiva su invocación al llegar al proceso principal.

DEVUELVE: Nada

EJEMPLO:

|FICHEROS;

datos #0;

|FIN;

|VARIABLES;

num\_tec = "";

|FIN;

|PROGRAMA;

num\_tec = " ";

[|PARA](#) ;[|SI](#) num\_tec < 1;[|O](#) num\_tec > 5; [|HACIENDO](#);

[|LEETECLA](#) num\_tec;

[|SI](#) num\_tec ] 1;[|Y](#) num\_tec [ 5;

|| instrucciones ...

[|FINSI](#);

```
|SI num_tec = 6;  
    |ABORTA;  
    |FINI;  
    || instrucciones ...  
    |SIGUE;  
|FPRC;
```

## ABRE

SINTAXIS: |ABRE #n;

DONDE:

n será el número de fichero definido en la parte referente a ficheros.

PROPÓSITO: Abre un fichero indexado. Si el fichero ya ha sido abierto la sentencia no tiene ningún efecto y tampoco devuelve error. Si el fichero no existe al abrirlo se intenta crear su indexado y, si esto tampoco resulta, entonces se devuelve el error en la variable reservada FSalida.

DEVUELVE:

FSalida = 0 Si ha ido bien la apertura.

FSalida > 0 Si se ha producido un error bien de indexado o del sistema operativo.

VER: |CIERRA |ABRET |CIERRAT

EJEMPLO:

```
|FICHEROS; || Declaración de ficheros.  
    faclien#0;  
|FIN;  
|PROCESO abrir;|TIPO 0;  
    |ABRE #0; || Abre fichero.  
    |SI FSalida = 0; || Estudio de resultado.
```



[|MENSA](#) "0000Fichero abierto"; || Pinta mensaje en  
(00,00)

[|SINO](#);

[|MENAV](#) "0000Fichero no se ha abierto";

[|FINSI](#);

[|CIERRA](#) #0; || Cierra fichero

|FPRC;

## ABRET

SINTAXIS: |ABRET #n;

DONDE: n será el número de fichero definido en la parte referente a ficheros.

PROPÓSITO: Abrir un fichero indexado. Un fichero abierto con |ABRET sólo se puede cerrar con su sentencia complementaria [|CIERRAT](#). Los ficheros abiertos por funciones o procesos del runtime (AGIRUN) se abren siempre de este modo. Si el fichero ya ha sido abierto la sentencia no tiene ningún efecto y tampoco devuelve error. Si el fichero no existe al abrirlo, se intenta crear su indexado y si esto tampoco resulta entonces se devuelve el valor correspondiente al error.

DEVUELVE:

FSalida = 0 Si la apertura ha sido correcta

FSalida > 0 En caso de error bien de indexado o del sistema.

VER: [|CIERRAT](#) [|ABRE](#) [|CIERRA](#)

EJEMPLO:

|FICHEROS;

facclien#0;

|FIN;

[|PROCESO](#) abrir;|TIPO 0;

[|ABRET](#) #0;

[|SI](#) FSalida = 0;

[|MENSA](#) "0000Fichero abierto";

```
|SINO;  
    |MENAV "0000Fichero no se ha abierto";  
  
|FINI;  
  
|HAZ suma;  
  
|CIERRAT #0;  
  
|FPRC;  
  
  
|PROCESO suma;|TIPO 0;  
  
|ABRE #0;  
  
|LEE 110#0=;  
  
|SI FSalida = 0;  
    #0#2 = #0#2 + importe;  
  
    |GRABA 020#0a;  
  
|FINI;  
  
|LIBERA #0;  
  
|CIERRA #0;  
  
|FPRC;
```

El cálculo suma ejecutado desde abrir no abre ni cierra el fichero pues el [|ABRET](#) tiene prioridad, sin embargo si se llamase desde otro proceso en el cual no estuviese abierto el fichero sí que tendría efecto el [|ABRE](#) y el [|CIERRA](#) de suma.

## ACABA

SINTAXIS: |ACABA;

PROPÓSITO: Finaliza la ejecución del proceso corriente.

DEVUELVE: Nada.

EJEMPLO:

|[PROCESO](#) Comprueba;|TIPO 0;

|[SI](#) #0#0> "123";

|[MENAV](#) "0400Valor NO permitido. Teclee otro.";

|[ERROR](#);

|[ACABA](#);

|[FINSI](#);

|FPRC

## AFEGIR\_FICHERO

SINTAXIS: |AFEGIR\_FICHERO alfa1, alfa2;

PROPÓSITO: Añade el fichero *alfa1* a *alfa2*.

Si FSalida < 0 error.

EJEMPLO:

|[PROCESO](#) afegir;

|[DBASE](#) dbse;

origen =dbase + "logs/parcial.dat";

destino = dbase + "logs/todo.dat";

|AFEGIR\_FICHERO origen, destino;

|FPRC;

## ALARMA

SINTAXIS: |ALARMA fecha, hora, aAlfa1;

DONDE:

Fecha es variable alfanumérica. En ella se indicará el día que se debe activar la alarma.

Hora es variable alfanumérica. Se indicará la hora de activación.

aAlfa1 es variable alfanumérica. Ejecutable que se cargará cuando la fecha y la hora se cumplan.

PROPÓSITO: Programa la alarma del sistema para la fecha y la hora en la que se ejecutará, en modo subordinado (debajo de la aplicación corriente si la hay) el ejecutable indicado en la variable aAlfa1. Después de la ejecución, la alarma se desactiva.

DEVUELVE: Nada.

VER: [|NOALARMA](#) [|SIALARMA](#)

EJEMPLO:

```
|FICHEROS;  
    agenda #0;  
|FIN;  
|VARIABLES;  
    fecha=@;  
    hora="12:45:00";  
    pro=":util/alarma.ds";  
|FIN;  
|PROCESO alarma1;|TIPO 8;  
    ALARMA fecha,hora,pro;  
|FPRC;
```

## ALFACALLDLL

SINTAXIS: |ALFACALLDLL aNomDLL, aAlfa1, aAlfa2;

DONDE:

aNomDLL es una variable alfanumérica que contiene el nombre de la dll.

aAlfa1 y aAlfa2 son variables alfanuméricas que contienen la información para acceder a los procedimientos públicos de una dll.

P.ej.

<u>aAlfa1</u>	<u>aAlfa2</u>	<u>Descripción</u>
"selecciona_hoja"	"nombre_hoja"	Selecciona la hoja excel
"poke_dato"	aAlfa	Mete datos en una celda, donde aAlfa = "Celda+"Valor""
"peek_dato"	aAlfa	Lee la celda aAlfa y guarda el contenido en aAlfa.
"fin_book"	aAlfa	Donde aAlfa puede ser: "imprime", "salva", "".

PROPÓSITO: Esta sentencia permite acceder a los procedimientos públicos de una dll.

DEVUELVE: Nada

VER: [|CARGADLL](#) [|DESCARGADLL](#)

EJEMPLO:

```
aAbre = "C:\CAJON\XDOCUMENTOS\OTP\XEVALUACION.XLS";
```

```
|CARGADLL "agixl97.dll";    ||Cargamos la dll
```

```
|SI FSalida < 0;
```

```
    |MENAV "0000No se puede usar agixl97.dll";
```

```
    |ACABA;
```

```
|FINSI;
```

```
|ALFACALLDLL "agixl97.dll", "carga_book", aAbre;    ||Abrimos el  
libro xevaluacion.xls
```

```
|ALFACALLDLL "agixl97.dll", "selecciona_hoja", "Evaluacion";  
||Seleccionamos la hoja Evaluación del libro xevaluacion.xls
```

```
|PARA; |SI; |HACIENDO;
```

```
aCampoA = "A" + nLinea;  
|ALFACALLDLL "agixl97.dll", "peek_dato", aCampoA;  
||Lee el dato de la celda aCampoA (p.ej. A1) y lo guarda en
```

||aCa  
mpoA

```
|QBF aCampoA;  
aCampoB = "B" + nLinea;  
|ALFACALLDLL "agixl97.dll", "peek_dato", aCampoB;  
|QBF aCampoB;  
nLinea = nLinea + 1;  
  
|SI nLinea ] 1024; |SAL; |FINSI;
```

[|SIGUE](#);

```
aComillas = "A";
```

```
aAlfa = "A1" + "," + aComillas + "PEPE" + aComillas;
```

```
|ALFACALLDLL "agixl97.dll", "poke_dato", aAlfa;    ||Introducimos  
"Pepe" en la celda "A1"
```

```
|ALFACALLDLL "agixl97.dll", "fin_book", "imprime";  
||Imprimimos
```

```
|ALFACALLDLL "agixl97.dll", "fin_book", "salva";    ||Guardamos
```

```
|ALFACALLDLL "agixl97.dll", "fin_book", "";  
||Cerramos el libro  
excel
```

```
|DESCARGADLL "agixl97.dll";    ||Descargamos la librería
```

## AL\_BUF

SINTAXIS: |AL\_BUF varnum1, varnum2, variable;

DONDE:

varnum1 es una variable numérica.

varnum2 es una variable numérica. Su rango está comprendido entre 0 y el máximo con que se ha dimensionado el buffer.

variable puede ser variable numérica, alfanumérica, tipo fecha o puntero.

PROPÓSITO: Esta sentencia ubica en el buffer varnum1 (que dependerá del valor devuelto por la sentencia |DIMENSIONA,), según la posición dada por varnum2 contenido de variable.

Hay que tener en cuenta que, en este caso, no existe conversión previa del tipo de variable, es decir si utilizamos un buffer de tipo fecha deberemos tratarlo íntegramente como tal a la hora de leerlo o grabar en él.

DEVUELVE: Nada

VER: |[DEL BUF](#) |[DIMENSIONA](#) |FINDIM

EJEMPLO:

```
|FICHEROS;

    prueba #0;

|FIN;

|VARIABLES;

    i = 0;

    bufter = 0;

    tmp = "";

|FIN;

|PROCESO albuffer;|TIPO 0;

|DIMENSIONA 233,bufter,0;

|SI bufter ] 0

    |AL BUF bufter,0,"";

|PARA i = 0;|SI i < 233;|HACIENDO i = i + 1;

    tmp = #0i;
```

```
|AL BUF bufter,i,tmp;  
  
|SIGUE;  
  
|| instrucciones ....  
  
|FINDIM bufter;  
  
|FINSI;  
  
|FPRC;
```

## ATAR

SINTAXIS: |ATAR aAlfa1, aAlfa2;

DONDE:

aAlfa1 = fichero destino(incluyendo path) + " " + lista de ficheros (separados por espacios)

aAlfa2 = directorio origen (acabado en /)

por ejemplo:

queremos empaquetar /u/ds/dscomer7/fich/01 en /tmp/copia.tar

aAlfa1 = "/tmp/copia.tar ."

aAlfa2 = "/u/ds/dscomer7/fich/01/"

otro ejemplo:

queremos empaquetar /u/ds/ds.men mas /u/ds/ds.ini en /tmp/confi.tar:

aAlfa1 = "/tmp/confi.tar ds.men ds.ini"

aAlfa2 = "/u/ds/"

## ATRI

SINTAXIS: |ATRI <letra>;

DONDE:

<letra>:      R -> activa reverse.



r -> desactiva reverse.

I -> activa intensidad.

i -> desactiva intensidad.

P -> activa parpadeo.

p -> desactiva parpadeo.

N -> atributo normal.

0 -> desactiva todos los atributos (atributo normal)

PROPÓSITO: Activa el atributo indicado en letra. Esta activación afectará a la presentación de todos los datos que tengan que aparecer en pantalla posteriormente.

DEVUELVE: Nada.

VER: [|PINTA](#) [|MENSA](#) [|MENAV](#) [|GRAF](#) |ATRIBUTOS\_DEFEC

EJEMPLO:

[|PROCESO](#) pintar;|TIPO 0;

[|ATRI](#) R;

|| Activa el reverse para los datos que le siguen.

[|PINTA](#) 1510,información;

[|ATRI](#) 0;

|| Desactiva todos los atributos hasta ahora activados para que no afecte a los datos posteriores.

|FPRC;

## **ATRIBUTOS\_DEFEC o ATRI\_DEF**

SINTAXIS: |ATRIBUTOS\_DEFEC cte1,cc1,cf1,cte2,cc2,cf2,cte3,cc3,cf3;

DONDE:

cte1, cte2, ct3 son constantes de carácter que pueden tomar los valores de atributo:

N - Normal.

I - Intensidad.

R - Reverse.

P - Parpadeo.

S - Subrayado.

cc1,cc2,cc3 son variables numéricas que corresponden al valor del COLOR de CARÁCTER.

cf1,cf2,cf3 son variables numéricas que corresponden al valor del COLOR de FONDO.

PROPÓSITO: Permite modificar los atributos y colores en la entrada y presentación de datos por teclado. Consta de tres grupos de tres valores cada uno.

El primer grupo permite asignar el ATRIBUTO y los colores de fondo y de carácter en el momento de presentar los datos de una ficha por pantalla.

El segundo grupo asignará los atributos y colores de los caracteres de la pantalla en el momento de introducción de datos.

El tercero grupo se utiliza para asignar atributos en la manipulación de campos tipo Texto o Memo.

Si cte1, cte2, cte3 tiene el valor D recogerá por defecto el ultimo valor que se le hubiera asignado, caso de que fuera la primera vez que se utiliza la instrucción, el valor seria el declarado por defecto en la CONFIGURACIÓN DE TERMINALES.

Tener en cuenta que ATRIBUTO\_DEFEC modifica los atributos durante TODA la aplicación, no solo en el programa donde se invoca.

DEVUELVE: Nada

VER: [|ATRI](#) [|PINTA](#) [|GRAF](#)

EJEMPLO:

|VARIABLES

cc1 = 0;

cc2 = 0;

cc3 = 0;

cf1 = 0;

cf2 = 0;

cf3 = 0;

|FIN;

|PROCESO cambia\_atributos

cc2 = 8;

cf2 = 7;

cc3 = 6;

cf3 = 1;

|ATRIBUTOS\_DEFECTO

D,cc1,cf1,N,cc2,cf2,N,cc2,cf2,N,cc3,cf3;

|| A partir de este momento cada vez que se presenten datos por pantalla

|| se harán con los parámetros por defecto generalmente en una pantalla

|| en color sería carácter amarillo sobre fondo azul

|| Cuando se introduzcan datos el atributo será carácter Negro intenso

|| sobre fondo blanco, generalmente es carácter azul sobre fondo blanco.

|| En la manipulación de campos Textos o memo el carácter será amarillo

|| sobre fondo azul, por defecto es Blanco sobre fondo negro.

|FPROC;

## AVISO

SINTAXIS: [|AVISO](#);

PROPÓSITO: Emite un sonido en el terminal de trabajo.

DEVUELVE: Nada.

EJEMPLO:

```
|PROCESO control;|TIPO 0;  
    |SI #0Campo ! Contenido;  
        |AVISO; || Pita emite un pitido el ordenador  
        #0Campo = Contenido;  
    |PINTA #0Campo;  
    |FINSI;  
|FPRC;
```

## BLANCO

SINTAXIS: [|BLANCO](#) <variable numérica>;

PROPÓSITO: Borra el contenido de la zona de pantalla comprendida entre la posición inicial y final contenidas en la variable numérica bajo la representación: ficiffcf.

DONDE:

fi es fila inicial, ci columna inicial, ff fila final y cf columna final.

DEVUELVE: Nada.

VER: [|PUSHV](#) [|POPV](#) [|CUADRO](#) [|CUADRO\\_D](#) [|CUADRO\\_S](#)

EJEMPLO:

```
|PROCESO cuadro;|TIPO 0  
    |BLANCO 0110 0640; || fi=01, ci=10, ff=06, cf=40  
    |CUADRO 0110 0640;  
|FPRC;
```

## BLIN

SINTAXIS: |BLIN n;

DONDE:

n es una variable numérica donde se indica el número de la línea a borrar.  
Puede tomar los valores desde 1 a 24.

PROPÓSITO: Borra una línea de la pantalla.

DEVUELVE: Nada.

VER: [|MENSA](#)

EJEMPLO:

```
||PROCESO borrarlin;|TIPO 0;  
    ||PARA i = 1;||SI i < 11;||HACIENDO i=i+1;  
        ||BLIN i; || Borra las 11 primeras líneas.  
    ||SIGUE;  
    ||BLIN 24; || Borra la línea 24  
|FPRC;
```

## BMENUG

SINTAXIS: |BMENUG vnum1,vnum2,vnum3,vnum4,vnum5,vnum6;

DONDE:

vnum1 es un buffer de tipo numérico donde estarán las opciones.

vnum2 principio o desde el nº de opción

vnum3 variable numérica que especifica el salto.(de 1 en 1, 2 en 2)

vnum4 fin o hasta el número de opción

vnum5 es una variable numérica que especifica la posición del menú según la representación.

ffcc, donde ff y cc son la fila y columna inicial. Si el valor es 0, el menú se centra.

vnum6 es variable numérica e indica la opción por defecto del menú.

PROPÓSITO: Presenta un menú tipo [|MENU|](#), en la posición dada por vnum5, cuyas opciones vendrán ubicadas en el buffer vnum1, desde vnum2 a vnum4 con un salto especificado en vnum3 ( de 1 en 1, de 2 en 2,... ) partiendo de la opción dada por vnum6.

DEVUELVE: Nada

VER: [|MENU|](#) [|MENU|](#)

EJEMPLO:

```
|FICHEROS;
    datos  #0;
|FIN;
|VARIABLES;
    arrai = 0;
    ndat = 0;
    menudat = "/Agi/dat/datos.sec";
|FIN;
|PROGRAMA;
    |LEESECUENCIAL menudat,arrai,ndat,0,0;
    |PARA; |SI ;|HACIENDO;
        |SI arrai ] 0;
            |SI FSalida = 7;|O FSalida = 1;
                |SAL;
            |FINSI;
        |BMENUG arrai,0,2,ndat,0,0;
        |SI FSalida < 1;
```

```
|SAL;  
  
|FINI;  
  
|| instrucciones ....  
  
|FINI;  
  
|SIGUE;  
  
|| instrucciones ...  
  
|FPRO;
```

## **BORRA**

SINTAXIS: |BORRA abc#nd;

DONDE:

'a' -> no usado, siempre a 0

'b' el flag de mensajes de error:

0-> solo hay mensajes de error en caso de bloqueo.

1-> errores fatales (fichero no abiertos ...)

2-> todos los mensajes.

4-> no da mensajes.

'c' flag de reintento:

0-> no reintentar en caso de error.

1-> reintentar sólo en caso de bloqueo

2-> reintentar siempre (con cualquier error) 'n' el número de fichero.

'd' modo de selección de la ficha a borrar:

a-> ficha actual (previamente leída)

c-> según la clave que indicamos.

PROPÓSITO: Borrar un registro de datos.

DEVUELVE: FSalida = 0 Ha ido bien

FSalida > 0 Error de indexado o del sistema.

VER: |[LEE](#)

EJEMPLO:

|[PROCESO](#) borra;|TIPO 0;

|[ABRE](#) #0;

|[LEE](#) 121#0=;

|[SI](#) FSalida = 0;|[Y](#) #0#10 = si;

|[BORRA](#) 020#0a;

|| Borra el registro si la lectura ha sido correcta y si el valor de #0#10 es 'si'.

|[FINSI](#);

|[LIBERA](#) #0;

|[CIERRA](#) #0;

|FPRC;

En este ejemplo, borrará mostrando todos los mensajes de error (b=2), no reintentando en caso de error de lectura (c=0) y eliminará la ficha actual (d='a').

## **BORRA\_HIJOS\_PAN**

SINTAXIS: |BORRA\_HIJOS\_PAN #numpan#fichero, idpantalla;

PROPÓSITO: Borra las pantallas hijas de la indicada (con todos sus controles).

## **BROWSE**

Muestra el diálogo abrir archivo, guardar archivo, abrir directorio. (A partir de la versión de básico 9.09V2). Esta instrucción trabaja en el cliente (en caso de ser una instalación cliente-servidor).

SINTAXIS: |BROWSE aAlfa1;



En aAlfa1:   aAlfa1 = "flag" + path;

Flag:

" " Abrir archivo

"S" Salvar archivo

"D" Abrir carpeta

en alfa(+1) vuelve el path o nada si se ha cancelado.

EJEMPLO:

Para guardar en aAlfa1 una carpeta que seleccione el usuario:

aAlfa1 = "D";

[|BROWSE](#) aAlfa1;

Para guardar en aAlfa1 un archivo que seleccione el usuario:

aAlfa1 = " " + "c:\\";

[|BROWSE](#) aAlfa1;

## BUCLE

SINTAXIS: |BUCLE n;

DONDE:

n es un número  $\geq 0$ ;

PROPÓSITO: Ejecutar una lectura de fichero definida en un DEFBUCLE. El número n corresponde al n-DEFBUCLE declarado siendo 0 el primero. Esta sentencia abre y cierra el fichero que trata. Habrá de tenerse en cuenta este detalle pues si previamente ha sido abierto con un simple [|ABRE](#) y no con un [|ABRET](#), tras la ejecución del [|BUCLE](#) el fichero se cerrará.

DEVUELVE:

VER: MANUAL DE PROCESOS BUCLES DE LECTURAS DE FICHEROS:  
BUCLE DECLARADO |DEFBUCLE [|ERROR](#) [|ERROR10](#)

EJEMPLO:

```
|PROCESO imprimir;|TIPO 0;  
|IMPRE 0;  
|SI FSalida ] 0;  
|INFOR informe;  
|SI FSalida ] 0;  
|BUCLE nombucle;  
|FINSI;  
|FININF;  
|FINSI;  
|FINIMP;  
|FPRC;
```

La sentencia '|BUCLE nombucle;' ejecutará la lectura del fichero ordenada y declarada en el primer '|DEFBUCLE;'.

## BUCLELIN

SINTAXIS: |BUCLELIN mnombreproceso#n;

DONDE: m es un flag de control de lectura, nombreproceso el nombre de un proceso definido en el ejecutable y n el número de un fichero con líneas.

PROPÓSITO: Lee las líneas del fichero de cabeceras n y para cada una ejecuta el proceso nombre proceso.

m determina el modo de leer las líneas :

m = 0 lee bloqueando.

m = 1 lee bloqueando borra las líneas si el proceso no tuviera ninguna operación, únicamente borraría

m = 2 lee sin bloquear.

Esta sentencia abre y cierra el fichero de líneas que trata. En caso de haberse abierto previamente con un `|ABRE` y no con un `|ABRET` el fichero se cerrará después del `|BUCLELIN`. Hay que tener en cuenta que, aun que el tratamiento de lectura se realice sobre el fichero de líneas, la sentencia debe ejecutarse sobre el fichero de cabeceras pues es él el que está asociado al fichero de líneas.

DEVUELVE:

FSalida = 0 Si la ejecución del bucle de líneas ha sido correcto.

FSalida = -1 En caso de error .

VER: MANUAL DE PROCESOS: BUCLE DE LECTURA DE FICHEROS.

`|ERROR`

EJEMPLO:

`|PROCESO` sumalin;|TIPO 0;

`#1#5 = cliente;`

`|GRABA` 020#1a;

`líneas = líneas + 1;`

`|FPRC;`

`|PROCESO` total;|TIPO 0;

`líneas = 0;`

`|BUCLELIN` 0sumalin#0;

`|| el fichero 1 es líneas del 0`

`|| y actualizamos un campo en todas las`

`|| líneas de la ficha del fichero 0`

`|| en líneas viene el total de líneas`

`|FPRC;`

**CABEZA**

SINTAXIS: |CABEZA <variable alfanumérica>;

PROPÓSITO: [Pinta](#) una cabecera estándar con el proceso contenido en la variable alfanumérica.

DEVUELVE: Nada.

VER: [|CLS](#) [|PINPA](#) [|MANTE](#)

EJEMPLO:

```
|PROGRAMA;  
  
    |CLS;  
  
    |CABEZA "ESTE ES EL TITULO";  
  
    |PINPA #0#0;  
  
    || ...  
  
|FPRO;
```

## **CAMPO\_ANCHO o C\_A**

SINTAXIS: |CAMPO\_ANCHO campo,valor, varnum;

DONDE:

campo corresponde a un campo con la representación standard #X#Y.

valor es una constante numérica que puede valer 0 o 1.

varnum es una variable numérica.

PROPÓSITO: Permite modificar la anchura del campo especificado en campo. Si valor = 0 la sentencia devolverá en varnum la anchura del campo y si valor=1, la sentencia recogerá la nueva anchura de varnum.

DEVUELVE: Nada.

EJEMPLO:

```
|FICHEROS;  
  
    clientes#0;  
  
|FIN;
```

```
|PROCESO set_campos;|TIPO 7;  
|SI #0#1 = 1;  
|CAMPO_ANCHO #0#3, 1 , 50;  
|SINO;  
|CAMPO_ANCHO #0#3, 1 , 30;  
|FINSI;  
|FPRC;
```

## **CAMPO\_ATRIBUTO**

SINTAXIS: |CAMPO\_ATRIBUTO #n#c, atri, cc, cf

DONDE:

n es el número de fichero

c es el número de campo que deseamos modificar su atributo.

atri es una constante de tipo carácter. Indicará el atributo que tendrá, tras ejecutarse la sentencia, el campo y podrá tomar los valores:

D - Defecto.

N - Normal.

I - Intensidad.

R - Reverse.

P - Parpadeo.

S - Subrayado.

cc, cf son los valores correspondientes al COLOR DE CARÁCTER y al COLOR DE FONDO respectivamente.

PROPÓSITO: Permite modificar los valores que, por defecto, tiene asignado el campo #n#c en su estructura interna. Si se indica en atri el valor D, tomará el color general por defecto modificable con ATRIBUTOS\_DEFEC y en caso contrario modificará su aspecto pero sin grabar dichos cambios en la estructura

interna con lo que todas las modificaciones deberán realizarse a través de procesos.

DEVUELVE: Nada

VER: [|ATRI](#) |ATRIBUTOS\_DEFEC [|PINTA](#)

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROCESO set_campos;|TIPO 7;  
  
    |SI #0#1 = 1;  
  
        |CAMPO\_ATRIBUTO #0#3, "R" , 1, 5;  
  
    |SINO;  
  
        |CAMPO\_ATRIBUTO #0#3, "P" , 1, 3;  
  
    |FINSI;  
  
|FPRC;
```

## **CAMPO\_LINEAL o C\_L**

SINTAXIS: |CAMPO\_LINEAL campo, valor, varalfa;

DONDE:

campo corresponde a un campo según su representación standard en formato #X#Y.

valor constante numérica de valores 0 o 1.

varalfa es una variable alfanumérica que podrá tomar los valores 'S' o 'N'.

PROPÓSITO: Permite modificar el FCD ( flag de control de datos) LINEAL del campo. Si valor es 0, en la variable varalfa recogeremos el estado del FCD Lineal y en caso de valor=1 podremos modificar el estado de dicho flag a través de varalfa.

DEVUELVE: Nada

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROCESO set_campos;|TIPO 7;  
  
    |SI #0#1 = 1;  
        |CAMPO LINEAL #0#3, 1 , "S";  
    |SINO;  
        |CAMPO LINEAL #0#3, 1 , "N";  
  
    |FINSI;  
  
|FPRC;
```

## **CAMPO\_MODIFICA o C\_M**

SINTAXIS: |CAMPO\_MODIFICA campo, valor, varalfa;

DONDE:

campo corresponde a un campo según su representación standard en formato #X#Y.

valor constante numérica de valores 0 o 1.

varalfa es una variable alfanumérica que podrá tomar los valores 'S' o 'N'.

PROPÓSITO: Permite modificar el FCD ( flag de control de datos) MODIFICABLE del campo. Si valor es 0, en la variable varalfa recogeremos el estado del FCD Modificable y en caso de valor = 1 podremos modificar el estado de dicho flag a través de varalfa.

DEVUELVE: Nada

EJEMPLO:

```

|FICHEROS;

    clientes#0;

|FIN;

|PROCESO set_campos;|TIPO 7;

    |SI #0#1 = 1;

        |CAMPO\_POSICION #0#3, 1 , 1550;

        |CAMPO\_MODIFICA #0#3, 1 , "S";

        |CAMPO\_VISUAL #0#3, 1 , "S";

    |SINO;

        |CAMPO\_POSICION #0#3, 1 , 1750;

        |CAMPO\_MODIFICA #0#3, 1 , "N";

        |CAMPO\_VISUAL #0#3, 1 , "N";

    |FINSI;

|FPRC;

```

## **CAMPO\_POSICION o C\_P**

SINTAXIS: |CAMPO\_POSICION campo, valor, varnum;

DONDE:

campo corresponde a un campo según su representación standard en formato #X#Y.

valor constante numérica de valores 0 o 1.

varnum es una variable numérica de posición, es decir con formato ffcc donde ff,cc son la fila y columna iniciales respectivamente.

PROPÓSITO: Permite cambiar la posición en pantalla de campo. Si valor = 0, en la variable varnum recogeremos su posición actual y si valor = 1 la sentencia cambiará la posición por la indicada en varnum.

DEVUELVE: Nada.



EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROCESO set_campos;|TIPO 7;  
  
    |SI #0#1 = 1;  
  
        |CAMPO_POSICION #0#3, 1 , 1550;  
  
        |CAMPO_MODIFICA #0#3, 1 , "S";  
  
        |CAMPO_VISUAL #0#3, 1 , "S";  
  
    |SINO;  
  
        |CAMPO_POSICION #0#3, 1 , 1750;  
  
        |CAMPO_MODIFICA #0#3, 1 , "N";  
  
        |CAMPO_VISUAL #0#3, 1 , "N";  
  
    |FINSI;  
  
|FPRC;
```

### **CAMPO\_VISUAL o C\_V**

SINTAXIS: |CAMPO\_VISUAL campo, valor, varalfa;

DONDE:

campo corresponde a un campo según su representación standard en formato #X#Y.

valor constante numérica de valores 0 o 1.

varalfa es una variable alfanumérica que podrá tomar los valores 'S' o 'N'.

PROPÓSITO: Permite modificar el FCD ( flag de control de datos) VISUALIZABLE del campo. Si valor es 0, en la variable varalfa recogeremos el

estado del FCD Visualizable y en caso de valor = 1 podremos modificar el estado de dicho flag a través de varalfa.

DEVUELVE: Nada

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROCESO set_campos;|TIPO 7;  
  
    |SI #0#1 = 1;  
  
        |CAMPO POSICION #0#3, 1 , 1550;  
  
        |CAMPO MODIFICA #0#3, 1 , "S";  
  
        |CAMPO VISUAL #0#3, 1 , "S";  
  
    |SINO;  
  
        |CAMPO POSICION #0#3, 1 , 1750;  
  
        |CAMPO MODIFICA #0#3, 1 , "N";  
  
        |CAMPO VISUAL #0#3, 1 , "N";  
  
    |FINSI;  
  
|FPRC;
```

## CARGADLL

SINTAXIS: |CARGADLL aAlfa;

DONDE: aAlfa es el nombre de la dll a cargar

PROPÓSITO: Permite cargar una dll

DEVUELVE: Nada

VER: [|ALFACALLDLL](#) [|DESCARGADLL](#)

EJEMPLO:

|CARGADLL "agixl97.dll";

## CDEFECTO

SINTAXIS: |CDEFECTO , #fichero, clave;

PROPÓSITO: Clave será el número de clave con la que estamos leyendo.

Clave principal sería 1.

Hace el defecto de toda la ficha excepto de los campos que forman parte de la clave, si la clave no es valida equivale a [DDEFECTO](#).

EJEMPLO:

|[PROCESO](#) lee; |TIPO 0

|[ABRE](#) #1;

#1#0=#0#0

|[LEE](#) 040 #1=;

|[SI](#) FSalida!0;

[CDEFECTO](#) #1#1;

|[FINSI](#)

|[CIERRA](#) #1;

|FPRC;

## CIERRA

SINTAXIS: |CIERRA #n;

DONDE:

n el número de fichero.

PROPÓSITO: Cierra el fichero n a menos que haya sido abierto con [ABRET](#) o por algún proceso de AGIRUN.

DEVUELVE:

FSalida = 0 Siempre que el cierre del fichero haya sido correcto.

FSalida > 0 Si se ha producido algún error al cerrarlo. Puede tratarse de errores de indexado o del sistema.

VER: [|ABRE](#) [|LEE](#) [|GRABA](#) [|BORRA](#)

EJEMPLO:

```
|FICHEROS;  
    facclien#0;  
|FIN;  
  
||PROCESO abrir;|TIPO 0;  
    ||ABRE #0;  
    ||SI FSalida = 0;  
        ||MENSA "0000Fichero abierto";  
    ||SINO;  
        ||MENAV "0000Fichero no se ha abierto";  
    ||FINSI;  
    ||CIERRA #0;  
|FPRC;
```

## CIERRAT

SINTAXIS: [|CIERRAT](#) #n;

DONDE:

n el número de fichero a cerrar.

PROPÓSITO: Cierra el fichero n de un modo absoluto. Es el homólogo de la sentencia [|ABRET](#) y se suele utilizar en los casos en que, tras la ejecución de otra sentencia, puede llegar a cerrarse el fichero sin desearlo. Entonces con ambos nos aseguramos que el fichero permanecerá abierto hasta que ejecutemos esta sentencia.

DEVUELVE:

FSalida = 0 Si el cierre ha sido correcto.

FSalida > 0 En caso de error de indexado o del sistema.

VER: [|ABRET](#) [|ABRE](#) [|CIERRA](#)

EJEMPLO:

|FICHEROS;

    facclien#0;

|FIN;

|[|PROCESO](#) abrir;|TIPO 0;

[|ABRET](#) #0;

[|SI](#) FSalida = 0;

[|MENSA](#) "0000Fichero abierto";

[|SINO](#);

[|MENAV](#) "0000Fichero no se ha abierto";

[|FINSI](#);

[|HAZ](#) suma;

[|CIERRAT](#) #0;

|FPRC;

|[|PROCESO](#) suma;|TIPO 0;

[|ABRE](#) #0;

[|LEE](#) 110#0=;

[|SI](#) FSalida = 0;

        #0#2 = #0#2 + importe;

[|GRABA](#) 020#0a;

[|FINSI](#);

|[LIBERA](#)#0;

|[CIERRA](#) #0;

|FPRC;

El proceso suma, ejecutado desde el proceso abrir, no abre ni cierra el fichero pues el [|ABRET](#) tiene prioridad, sin embargo si se llamase desde otro proceso en el cual no estuviese abierto el fichero sí que tendría efecto el [|ABRE](#) y el [|CIERRA](#) de suma.

## CIMPR

SINTAXIS: |CIMPR n;

DONDE:

n un número comprendido entre 0 y 10.

PROPÓSITO: Envía a la impresora el carácter de control n según definición del fichero de la impresora seleccionada.

Los valores posibles de n a enviar son los siguientes:

10 = devuelve en FSalida ancho de la impresora

0 = desactiva todos los atributos

1 = activar comprimido

2 = desactivar comprimido

3 = activar expandido

4 = desactivar expandido

5 = activar negrita

6 = desactivar negrita

11 = devuelve en FSalida el número de líneas de la impresora seleccionada.

DEVUELVE:

FSalida = 0 si la operación ha sido correcta.

FSalida = -1 si se ha producido algún error.

FSalida = 10 devuelve el ancho de la impresora.

VER: [|IMPRE](#) [|FINIMP](#) [|INFOR](#)

EJEMPLO:

||Si la impresora es de menos de 132 ( es de 80 ) activa comprimido.

[|PROCESO](#) imprime;|TIPO 0;

[|IMPRE](#) 0;

[|SI](#) FSalida ] 0;

[|CIMPR](#) 10; || Comprueba el ancho de carro.

[|SI](#) FSalida < 132;

[|CIMPR](#) 1; || Activa comprimido.

[|FINSI](#);

[|CIMPR](#) 0;|| Desactiva todo atributo.

[|FINSI](#);

[|FINIMP](#);

|FPRC;

## CLS

SINTAXIS: |CLS;

PROPÓSITO: Borrar la pantalla.

DEVUELVE: Nada.

VER: [|PINPA](#) [|MANTE](#) [|CABEZA](#) [|BLIN](#)

EJEMPLO:

|PROGRAMA;

[|CLS](#); || Borra la pantalla.

|[CABEZA](#) "ESTE ES EL TITULO";

|[PINPA](#) #0#0;

...

|FPRO;

## COLOR

SINTAXIS: |COLOR varnum1, varnum2;

DONDE:

varnum1 es una variable numérica que define el color del Carácter.

varnum2 es una variable numérica que define el color de Fondo.

PROPÓSITO: Define el color de los caracteres impresos en pantalla, a través de las sentencias propias de impresión como |[PINTA](#), |[CUADRO](#),.... El color se activará después de la invocación de la sentencia |[ATRI](#) que permitirá activar los atributos propios del carácter. Para anular el efecto del color volveremos a ejecutar la sentencia |[ATRI](#).

Tanto los colores de fondo como de carácter se codifican desde 0 a 7 según la tabla:

0 = NEGRO

1 = AZUL

2 = VERDE

3 = CÍAN

4 = ROJO

5 = MAGENTA

6 = AMARILLO

7 = BLANCO

y, además, los colores de carácter pueden codificarse desde 8 a 15 con la misma secuencia anterior pero con la característica adicional de INTENSIDAD. Ahora bien si, utilizando la sentencia |[ATRI](#), hemos activado previamente la



característica de INTENSIDAD, aunque tomemos un color perteneciente al rango 8-15 se ignora el color.

DEVUELVE: Nada

EJEMPLO:

```
|FICHEROS;  
    terminal#0;  
  
|FIN;  
  
|VARIABLES;  
    j = 0;  
    h = 0;  
  
|FIN;  
  
|PROCESO set_color;|TIPO 0;  
    j = #0#1;  
    h = #0#2;  
  
    |COLOR j,h;  
  
|FPRC;
```

## COMPRIME

SINTAXIS: |COMPRIME fichero; (Sin extensión)

PROPÓSITO: Comprime el fichero especificado.

VER: [|DESCOMPRIME](#), [|ATAR](#), [|DETAR](#)

EJEMPLO:

```
|PROCESO comprime;  
    direc="";  
  
    |DIRECTORIO direc;  
  
    fichero=direc+"logs/pepe";
```

|COMPRIME fichero;  
|FPRC;

## CONFI

SINTAXIS: |CONFI v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Como [|MENAV](#) pero espera que se pulse S o N cuyo valor por defecto viene indicado después de la posición.

DEVUELVE:

FSalida = 0 -> Se ha pulsado S.

FSalida = -1 -> Se ha pulsado N.

VER: [|MENSA](#) [|MENAV](#) [|PAUSA](#)

EJEMPLO:

[|CONFI](#) "2400NEs correcto? ";

Resultará en la pregunta centrada en la línea 24:

Es correcto? N(por defecto).

## CONSULTA\_CLAVE

Sintaxis: |CONSULTA\_CLAVE #clave#def;

Saca todo el contenido del fichero deseado por la clave seleccionada.

Los parámetros que se le pasan son:

#clave#def -> Número de clave y def del que queremos obtener la consulta.

Ejemplo:

[|CONSULTA\\_CLAVE](#) #4#dsimpm31;

## CONSULTA\_F\_CLAVE

Sintaxis: |CONSULTA\_F\_CLAVE #clave#def, proceso\_min , proceso\_max;

Los parámetros que se le pasan son:

#clave#def -> Número de clave y def del que queremos obtener la consulta.

proceso\_min ,

proceso\_max. Son dos procesos que nos sirven para colocar los límites de la consulta.

Ejemplo:

|[PROCESO](#) Min0;

#dsimp31#4 = "S";

#dsimp31#0 = " ";

|FPRC;

|[PROCESO](#) Max0;

#dsimp31#4 = "S";

#dsimp31#0 = "zzzzzzzz";

|FPRC;

|[CONSULTA\\_F\\_CLAVE](#) #4#dsimp31, Min0, Max0;

## COPIA\_FICHERO

SINTAXIS: |COPIA\_FICHERO alfa1, alfa2;

PROPÓSITO: Copia el fichero alfa1 a alfa2.

Si FSalida < 0 error.

EJEMPLO:

|[PROCESO](#) copiar;

|[DFICE](#) dfich;

origen=dfich + "origen.dat";

destino=dfich+"destino.dat";

|[COPIA\\_FICHERO](#) origen,destino;

|FRPC;

## CORRE

SINTAXIS: |CORRE varalfa;

DONDE: varalfa es una variable alfanumérica que contiene el proceso a ejecutar.

PROPÓSITO: Ejecuta un programa ejecutable contenido en varalfa como si se llamase desde menú. Su ejecución sólo es efectiva en caso de invocase desde un proceso principal o cuando se finaliza la ejecución del fichero ejecutable.

DEVUELVE: Nada.

EJEMPLO:

|PROGRAMA

.....

.....

|[CORRE](#) = "program1";

|FRPC;

## CUADRO

SINTAXIS: |CUADRO <variable numérica>;

PROPÓSITO: Pintar un cuadro gráfico comprendido entre la posición inicial y final contenidas en la variable numérica cuya representación es: ficiffcf donde fi es fila inicial, ci columna inicial, ff fila final y cf columna final.

DEVUELVE: Nada.

VER: |[BLANCO](#) |[GRAF](#) |[CUADRO\\_D](#) |[CUADRO\\_S](#)

EJEMPLO:

|[CUADRO](#) 01100640;

||DONDE 01 -> fi, 10 -> ci, 06 -> ff, 40 -> cf.

|| Pintaría el cuadro así

## **CUADRO\_D**

SINTAXIS: |CUADRO\_D varnum1,varnum2.

DONDE :

varnum1 = posición inicial.

varnum2= posición final.

PROPÓSITO: Pinta un cuadro doble.

DEVUELVE : Nada.

Ejemplo :

|CUADRO\_D 0110,0640;

## **CUADRO\_S**

SINTAXIS: |CUADRO\_S varnum1,varnum2.

DONDE :

varnum1 = posición inicial.

varnum2= posición final.

PROPÓSITO: Pinta un cuadro simple como [|CUADRO.](#)

DEVUELVE : Nada.

Ejemplo :

|[CUADRO\\_S](#) 0110,0640;

## **DAREGN**

SINTAXIS: |DAREGN #n, varocamnum;

DONDE:

n corresponde al número de fichero.

varocamnum es una variable o campo ( #X#Y )

PROPÓSITO: Conocer el número de registro a que corresponde la última ficha leída o grabada según clave.

DEVUELVE: En la variable varocamnum el número de registro actual.

EJEMPLO:

[|DAREGN](#) #3,varRegis;

Devuelve en varRegis el número de registro en que se encuentra el fichero #3.

## **DBASE**

SINTAXIS: |DBASE v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Permite obtener el directorio base más el de la aplicación.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#) [|DBASS](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest.

[|DBASE](#) dirbase; en dirbase devuelve '/Agi/gest'.

## **DBASS**

SINTAXIS: |DBASS v;

DONDE: v es una variable alfanumérica

PROPÓSITO: Permite obtener el directorio base.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#) [|DBASE](#)

EJEMPLO:

Caso de un directorio base /Agi.

[|DBASS](#) dirbase; en dirbase devuelve '/Agi/'

## **DBIN**

SINTAXIS: |DBIN v;

DONDE: v es una variable alfanumérica

PROPÓSITO: Permite obtener el path del directorio de los bin, o ejecutables, de nuestra aplicación.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#) [|DBASE](#) [|DBASS](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest.

[|DBIN](#) dirbin; en dirbin devuelve '/Agi/gest/bin/'.

## **DDEF**

SINTAXIS: |DDEF v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Permite obtener el directorio de los def.

DEVUELVE: Nada.

VER: [|INFOR](#) [|PATH\\_DAT](#) [|PATH\\_PAN](#) [|DBASE](#) [|DBASS](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest.

[|DDEF](#) dirdef; En dirdef devuelve '/Agi/gest/def/'

## **DDEFECTO**

SINTAXIS: |DDEFECTO #n;

DONDE: n un número de fichero.

PROPÓSITO: Pone en los campos del fichero n sus valores de defecto definidos en el def.

DEVUELVE: Nada.

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROCESO entrar;|TIPO 0;  
    |DDEFECTO #0;  
    |ENDATOS #0#1;  
  
|FPRC;
```

## DEBUG

SINTAXIS: |DEBUG;

PROPÓSITO: Activa el TRACER automáticamente en un punto determinado. Se Utiliza para poner puntos de depuración en unos puntos determinados del programa.

DEVUELVE: Nada.

EJEMPLO:

```
|PROCESO ejemplo;|TIPO 0;  
    |LEE 020#0=;  
    #0#5 = #0#5 + importe;  
    |GRABA 020#0a;  
    |DEBUG;  
    |CIERRA #0;
```



|FPRC;

## DEFECTO

SINTAXIS: |DEFECTO #n#f;

DONDE: f el número del fichero relacionado con n (también número de fichero).

PROPÓSITO: Recoger los valores de defecto definidos en el def del fichero n respecto del f.

DEVUELVE: Nada.

VER: |RDEFECTO

EJEMPLO:

|FICHEROS:

clientes#0;

comisión#1;

|FIN;

|[PROCESO](#) defecto;|TIPO 0;

|[DEFECTO](#) #0#1;

|FPRC;

## DELINDEX

SINTAXIS: |DELINDEX #n;

DONDE: n un número de fichero.

PROPÓSITO: Borra físicamente del disco el fichero de datos n y su índice.

DEVUELVE:

FSalida = 0 Si la operación de borrado se ha efectuado correctamente.

FSalida > 0 Si no ha sido posible borrar del disco el fichero de datos y su índice debido a un error de indexado o del sistema.

VER: [|ABRE](#)

EJEMPLO:

```
|FICHERO;  
    clientes#0;  
  
|FIN;  
  
|PROCESO borra;|TIPO 0;  
  
    |DELINDEX #0;  
  
|FPRC;
```

## DEL\_BUF

SINTAXIS: [|DEL\\_BUF](#) varnum1, varnum2, variable;

DONDE:

varnum1 es una variable numérica que representa al buffer con que se trabaja.

varnum2 es una variable numérica que representa la posición dentro del buffer.

variable es una variable genérica, bien alfanumérica, numérica, fecha o puntero.

PROPÓSITO: Ejecuta la operación contraria a la efectuada por la sentencia [|AL\\_BUF](#), recogiendo del buffer varnum1 en la posición varnum2 la variable.

DEVUELVE: Nada.

VER: [|AL\\_BUF](#) [|DIMENSIONA](#) [|FINDIM](#)

EJEMPLO:

```
|FICHEROS;  
    datos #0;  
  
|FIN;  
  
|VARIABLES;
```

```

        arrai = 0;

        nter = 0;

        tmp = "";

|FIN;

|PROGRAMA;

        || instrucciones ...

        |BMENUG arrai,0,2,nter,0,0;

        |SI FSalida < 1;

                |SAL;

        |FINSI;

        |SI FSalida ! 1;

                i = FSalida - 1;

                i = i * 2 + 1;

                |DEL BUF arrai,i,tmp;

        |FINSI;

        || instrucciones ...

|FPRO;

```

## DESCARGADLL

SINTAXIS: |DESCARGADLL aAlfa

DONDE: aAlfa es una variable alfanumérica que contiene el nombre de la dll a descargar.

VER: [|ALFACALLDLL](#) [|CARGADLL](#)

PROPÓSITO: Descarga una dll.

EJEMPLO:

```
|DESCARGADLL "agixl97.dll";
```

## DESCOMPRIME

SINTAXIS: |DESCOMPRIME fichero; (Sin extensión)

PROPÓSITO: Descomprime un fichero anteriormente comprimido.

EJEMPLO:

```
|PROCESO descomprime;  
    direc="";  
    |DIRECTORIO direc;  
    fichero=direc+ "tras/aplica.Z"  
    |DESCOMPRIME fichero;  
|FRPC;
```

## DESTRUYECONTROL

SINTAXIS: |DESTRUYECONTROL hadle;

PROPÓSITO: Se le pasa el hadle del control a destruir (numérico), por ejemplo un grid.

## DETAR

SINTAXIS: |DETAR fichero.tar, dir\_base;

PROPÓSITO: Extrae los ficheros contenidos en un archivo tar, dir\_base es el directorio destino.

EJEMPLO:

```
|DETAR "c:\temp\archivo.tar","c:\temp";
```

## DFICB

SINTAXIS: |DFICB v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Permite obtener el directorio de empresa.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest.

|DFICB diremp; en diremp vuelve '/Agi/gest/fich/'

## **DFICE**

SINTAXIS: |DFICE v;

DONDE: v es una variable alfanumérica

PROPÓSITO: Permite obtener el directorio de ficheros.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest y una empresa seleccionada 01

[|DFICE](#) dirfic; en dirfic vuelve '/Agi/gest/fich/01/'

## **DIMENSIONA**

SINTAXIS: |DIMENSIONA varnum1, varnum2, varnum3;

DONDE:

varnum1 es una variable numérica que contiene el número máximo de elementos del buffer.

varnum2 es una variable numérica que devuelve un número que caracteriza al buffer creado.

varnum3 es una variable numérica que indica qué tipo de buffer ha sido creado, si alfanumérico, numérico, fecha, .. con la siguiente codificación:

0 - Alfanumérico.

3 - Numérico.

5 - Fecha.

PROPÓSITO: Dimensiona un buffer interno o array teniendo en cuenta el tamaño del buffer a crear, o varnum1, y su tipo dado por varnum3.

DEVUELVE: En varnum2 el número asignado al buffer dimensionado.

FSalida > 0 Número característico del buffer

FSalida = -1 En caso de no poderse dimensionar presenta un error.

VER: [|AL\\_BUF](#) [|DEL\\_BUF](#) [|FINDIM](#)

EJEMPLO:

[|FICHEROS](#);

prueba #0;

[|FIN](#);

[|VARIABLES](#);

i = 0;

bufter = 0;

tmp = "";

[|FIN](#);

[|PROCESO](#) albuffer;[|TIPO](#) 0;

[|DIMENSIONA](#) 233,bufter,0;

[|SI](#) bufter ] 0

[|AL\\_BUF](#) bufter,0,"";

[|PARA](#) i = 0;[|SI](#) i < 233;[|HACIENDO](#) i = i + 1;

tmp = #0i;

[|AL\\_BUF](#) bufter,i,tmp;

[|SIGUE](#);

|| instrucciones ....

[|FINDIM](#) buffer;

[|FINSI](#);

|FPRC;

## DIRECTORIO

SINTAXIS: |DIRECTORIO aAlfa1;

DEVUELVE: el directorio base donde esta instalado el básico.

## DPAN

SINTAXIS: |DPAN v;

DONDE:

v es una variable alfanumérica.

PROPÓSITO: Permite obtener el directorio de las pantallas.

DEVUELVE: Nada.

VER: [|PATH\\_DAT](#) [|PATH\\_PAN](#)

EJEMPLO:

Caso de un directorio base /Agi y de una aplicación llamada gest.

[|DPAN](#) dirpan; en dirpan vuelve '/Agi/gest/pan/'

## DSCORREO

Sirve para enviar y recibir correos electrónicos apoyándose en un archivo externo llamado dscorreos.exe (funciona a partir del Diagram9.09N).

### |DSCORREO\_ENVIA

[ip\\_servicio](#) [dscorreos](#),[servidor\\_correo](#),[from](#),[to](#),[subject](#),[mensaje](#),[ficheros](#);

ip\_servicio -> alfa

- ip o nombre(dns) del host  
donde está corriendo el servicio  
[dscorreos](#) (ver 2.9.2 o superior!)

servidor \_ correo -> alfa

- ip o nombre(dns) del host de Internet que gestiona el servicio smtp o "MAPI" para usar un cliente "Outlook" como intermediario.

- Si se pone "LLAMA" dscorreos busca entre las variables de entorno una que se llame DS\_LLAMA y ejecuta lo que encuentre. Si se pone "CUELGA" dscorreos busca entre las variables de entorno una que se llame DS\_CUELGA y ejecuta lo que encuentre.

from -> alfa

- dirección de e-mail del origen del correo (sólo una!).

to -> alfa

- dirección e-mail de destino del correo, si son varios se separan con ";

subject -> alfa

- texto que aparecerá como "sujeto" del mensaje

mensaje -> alfa

- texto (una línea) que aparecerá en el cuerpo del mensaje.

ficheros -> alfa

- ficheros a incluir como "attachment". dscorreos 2.9.2 debería soportar varios pero sólo se ha probado extensivamente con uno.

Si FSalida = -1

- no se ha enviado

Si a la dirección de origen le anteponemos un símbolo '!', el correo se envía con un flag de confirmación de lectura.



Si lo que queremos hacer es enviar un correo autenticando la cuenta de envío, lo que indicaremos en la dirección origen es :

Contraseña de la cuenta de correo a autenticar.

El símbolo ':' Usuario de la cuenta de correo a autenticar.

El símbolo ':' Dirección de correo de origen.

P.Ej. :

Cuenta de correo : [pepe.lopez@servidor.es](mailto:pepe.lopez@servidor.es)

Usuario ..... : User

Contraseña ..... : Pwd

Indicaríamos como dirección de origen la siguiente cadena :

[Pwd:User:pepe.lopez@servidor.es](mailto:Pwd:User:pepe.lopez@servidor.es)

## **|DSCORREO\_RECIBE**

[ip\\_servicio\\_dscorreoservidor\\_correo\\_usuario\\_password\\_path\\_ficheros;](#)

ip\_servicio -> alfa

- ip o nombre(dns) del host donde esta corriendo el servicio dscorreos (ver 2.9.2 o superior!)

servidor \_ correo -> alfa

- ip o nombre(dns) del host de Internet que gestiona el servicio pop3 o "MAPI" para usar un cliente "Outlook" como intermediario. Si se pone "LLAMA" dscorreos busca en el entorno DS\_LLAMA y ejecuta lo que encuentre. Si se pone "CUELGA" dscorreos busca en el entorno DS\_CUELGA y ejecuta lo que encuentre.

usuario -> alfa

- usuario de la cuenta de correo

password -> alfa

- password del usuario de la cuenta de correo

path -> alfa	- path (local) donde se volcaran los ficheros recibidos como "attachment"
ficheros -> alfa	- lista (solo los nombres) de los recibidos (separados por ;).
Si FSalida = 1	- no hay mensajes en el servidor de correo
Si FSalida = -1	- no se ha recibido nada, fallo del sistema de correo

## EDITA

SINTAXIS: |EDITA vn1, vn2, vn3, vn4, vn5, vn6, valfa, vn7, vn8, vn9

DONDE:

vn1 es una variable numérica que indica el número de buffer a editar.

vn2 es una variable numérica que junto a vn3 marcarán los límites de lectura de líneas del buffer.

vn4, vn5 variables numéricas que delimitan la presentación en pantalla del editor.

vn6 variable numérica que representa el modo de edición donde son posibles los siguientes valores:

0 - Entra a editar

1 - No edita, sólo visualiza.

2 - Visualiza y si se pulsa Return entra a editar.

valfa es una variable alfanumérica donde se indicará un posible título que deseemos incluir en el editor.

vn7 es una variable numérica donde especificaremos la anchura que poseerá el editor. El valor máximo permitido es de 256 caracteres.

vn8 es una variable numérica donde detallaremos el número de línea inicial por defecto de la edición.

vn9 es una variable numérica que representará el número de columna inicial por defecto de la edición. Va asociada a la variable vn8.

PROPÓSITO: Permite editar cualquier texto que se desee, dependiendo del modo elegido, almacenado en el buffer vn1 y delimitado en pantalla por vn4 y vn5. El texto editado permanecerá en memoria y a través de vn2 y vn3 podremos determinar qué cantidad de líneas editaremos.

DEVUELVE: Nada

VER: [|F\\_TEXTO](#)

EJEMPLO:

[|PROCESO](#) edita;

[|DIMENSIONA](#) 233,vn1,0;

vn2=0;

vn3=0;

vn4=0701

vn5=2380;

vn6=0;

vn7=130;

vn8=0;

vn9=0;

[|EDITA](#) vn1,vn2,vn3,vn4,vn5,vn6,vn7,vn8,vn9;

[|SI](#) FSalida!0;

[|FINDIM](#) vn1;

[|ACABA](#);

[|FINSI](#);

[|FINDIM](#) vn1;

[|FPRC](#);

## ENCAMPO

SINTAXIS: |ENCAMPO #n#f;

DONDE: f el número de fichero y n el número de campo. El modo de entrada siempre será alta

PROPÓSITO: La entrada de un campo según está definido en el def y con sus mismo atributos.

DEVUELVE:

FSalida = número de control usado para salir del campo.

0 = Return

1 = Escape

2 = Flecha Arriba

3 = Flecha Abajo

4 = Página Arriba

5 = Página Abajo

6 = Consulta

7 = Abortar (CTRL-C)

8 = Ayuda (HELP)

9 = Función 1

10 = Función 2

11 a n = Función 3 a n (hasta 18 funciones)

VER: |[ENDATOS](#) |[PAUSA](#) Operación ?

EJEMPLO:

|FICHEROS;

clientes #0;

|FIN;

|[PROCESO](#) entrar;|TIPO 0;

|[ENCAMPO](#) #5#0; || Alta del campo 5 de clientes.

|FPRC;

## ENCLAVE

SINTAXIS: |ENCLAVE #fichero, clave, modo;

PROPÓSITO: Hace el "[ENDATOS](#)" de la clave tal y como lo hace el runtime en el caso de alta modif o consulta.

DONDE:

modo -> numérica

modo = 1 alta

modo = 2 modif

modo = 3 baja

modo = 4 consulta

clave -> numérica

numero de la clave (a partir de 0)

## ENDATOS

SINTAXIS: |ENDATOS #n#f;

DONDE:

f el número de fichero y n el modo de entrada de datos:

1 = Modo alta. En modo alta todas las relaciones, entre ficheros, se leen al pasar por el campo la primera vez.

2 = Modo modificación.

3 = Modo Baja.

4 = Modo consulta. No se leen las relaciones.

PROPÓSITO: La entrada de datos del fichero f según se ha definido en su mantenimiento. Esta sentencia nos permitirá ejecutar modos de mantenimiento

independientes en un fichero pues puede que, en algún momento, nos interese separar estos modos según las condiciones presentes.

DEVUELVE:

FSalida = 0 Si los datos ha sido aceptados,

FSalida = -1 Si los actos introducidos no son válidos.

VER: [|MANTE](#) [|ENCAMPO](#) [|PINPA](#) [|PINDA](#)

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
  
|FIN;  
  
|PROGRAMA;  
    |CLS;  
    |CABEZA "ESTE ES EL TITULO";  
    |PINPA #0#0;  
    |ENDATOS #1#0; || Alta registro clientes.  
  
|FPRO;
```

## ENTCOLUMNA

Igual que el [ENTLINEAL](#) pero va por columnas, el límite no es la línea sino el número de columnas.

## ENTLINEAL

SINTAXIS: [|ENTLINEAL](#) #c#n, vnum1, vnum2, vnum3, vnum4, proc1, proc2

DONDE:

c Número de clave a usar.

n Número de fichero a tratar.

vnum1 variable numérica. Corresponde a uno de los campos de la clave a usar como referencia de la línea, teniendo en cuenta que vnum1=1 será el primer campo,...

vnum2 variable numérica. Modo de entrada según convención standard ( 1 = alta, 2 = modificación, 3 = baja y 4=consulta )

vnum3 variable numérica. Número de línea límite de la entrada lineal.

vnum4 variable numérica. Flag que puede tomar los valores 0 ó 1 donde 0 = Equivale a no pintar la propia pantalla de líneas ( el fichero .pan) y al salir no borrar las líneas.

1 = Lo contrario, pintar la pantalla y al salir borrar las líneas.

proc1 variable alfanumérica que invoca un proceso donde asignaremos los valores del límite INFERIOR a cada componente de la clave elegida para la entrada lineal.

proc2 variable alfanumérica que invoca un proceso donde asignaremos los valores del límite SUPERIOR a cada componente de la clave elegida para la entrada lineal.

EJEMPLO:

```
|FICHEROS;  
    datos #0;  
|FIN;  
|ROUTINA inferior;  
    #0#0 = 0;  
    #0#2 = 0;  
|FRUT;  
|ROUTINA superior;  
    #0#0 = 23;  
    #0#2 = 59;  
|FRUT;
```

|PROGRAMA;

|[PINPA](#) #0#0;

|[ABRET](#) #0;

|[ENTLINEAL](#) #2#0,3,1,20,0,inferior,superior;

|[CIERRAT](#) #0;

|FPRO;

Notas a [|ENTLINEAL](#) ...;

La presentación es modo scroll, además si se llaman desde [ENTLINEAL](#) en el proceso donde se define el límite inferior se llama dos veces una de las cuales con la variable FSalida conteniendo el valor 'POSICION', entonces en vez del límite inferior se puede definir la posición (mediante el valor correspondiente de la clave) en que se situarán las líneas al entrar.

Nueva funcionalidad:

Modo alta: el orden de las líneas no se respeta, se mantienen tal como se van entrando hasta que se retrocede o se cambia de modo, en ese momento se reordena la presentación. En el campo control de la línea la tecla **Inicio** salta a la primera y la **fin** a la última.

Modo consulta-modificación: Cambiar el código de control produce su búsqueda situándose en el igual o superior.

Nuevos Flags a la variable Control: (usar |PONCONTROL para poner flags a esta variable dce sistema)

Posición 35: "MAS" indica al modo alta que se han grabado líneas nuevas, produce el repintado y la continuación el alta.

Este flag se 'resetea' en el momento en que actúa.

Posición 35: "INS" indica al modo modificación que se debe pasar a modo alta insertando líneas en la posición corriente.



Este flag se 'resetea' en el momento en que actúa.

El modo inserción viene anulado al pasar a modo modificación por acción del usuario.

Posición 33: "xx" donde en xx se ponen el número de líneas a desplazar para la inserción provocada con "INS".

Posición 30: "NOR" No repinta la línea corriente en modo alta. Evita efectos de repintado duplicado o que se pinten datos no deseados para una posible alta.

Posición 23: "si" repinta las líneas. Puede pasar a modo alta

Posición 23: "SI" repinta las líneas. No puede pasar a modo alta (localización).

|LINEAS\_BI\_ATRI Campo,Atributo1,Atributo2,Separador;

Esta instrucción hace que las líneas se pinten con diferente atributo en función del cambio de valor del campo indicado. Al cambiar de valor Campo se van usando alternativamente atributo1 y atributo2. Separador es el carácter usado para separar las líneas. Estas deben tener los campos comenzando detrás del separador, esto es para acelerar el repintado y evitar el efecto visual de repintar las columnas del campo.

|OCULTA\_LINEAS Fichero,varnum;

Si varnum = 1 se desvinculan las líneas de Fichero.

Si varnum = 9 se vuelven a vincular las líneas de Fichero.

Explicación: Si Fichero tiene líneas éstas se pintan y mantienen como dependencia de Fichero, con esta instrucción se puede hacer que Fichero se mantenga como si no tuviera líneas.

## **ENVIA\_FICHERO**

SINTAXIS: |ENVIA\_FICHERO aAlfa1, aAlfa2;

PROPÓSITO: Copiará el fichero aAlfa1 situado en la máquina del servidor en aAlfa2 situándose en el cliente.

## ERROR

SINTAXIS: |ERROR;

PROPÓSITO: Variado: entrando un campo hace que éste no sea válido y se deba volver a entrar, en un [BUCLELIN](#) provoca el abandono del mismo, y en un BUCLE hace que la ficha corriente no sea válida.

DEVUELVE: Nada.

VER: |[BUCLELIN](#) |BUCLE |[ENCAMPO](#)

EJEMPLO:

```
|FICHEROS;  
    valor #0;  
  
|FIN;  
  
|VARIABLES;  
    valor= 1;  
  
|FIN;  
  
|PROCESO control;|TIPO 0;  
    |SI #0Campo ! valor;  
        |AVISO;  
        |ERROR; || campo no valido  
        |ACABA;  
    |FINSI;  
  
|FPRC;
```

## ERROR10

SINTAXIS: |ERROR10;

PROPÓSITO: Provoca el abandono dentro de un BUCLE.

DEVUELVE: Nada.

VER: |BUCLE

EJEMPLO:

[|PROCESO](#) enbucle;|TIPO 0;

[|SI](#) #0#1 ] ultima;

[|ERROR10](#); || El proceso ENBUCLE es llamado desde bucle.

[|FINSI](#);

|FPRC;

## ESPECIFICA

SINTAXIS:

PROPÓSITO:

SALIDA:

EJEMPLO:

|FICHEROS;

md5;

|FIN;

|VARIABLES;

{1}md5 = "";

handle = 0;

|FIN;

|PROCESO browse;|TIPO 7;

md51 = "F";

[|BROWSE](#) md51;

#md5#0 = md51%2;

|FPRC;

|PROCESO md5;|TIPO 0;

md51 = #md5#0;

|[QBF](#) md51;

|ESPECIFICA "MD5",md5;

#md5#1 = FSalida;

|[PINTA](#) #md5#1;

|FPRC;

|PROGRAMA;

md51 = PARAMETRO;

|[QBF](#) md51;

|ESPECIFICA "MD5",md5;

md51 = FSalida;

handle = -1;

|[XABRE](#) "W","./pepe.log",handle;

|[XGRABA](#) handle,md51;

|[XCIERRA](#) handle;

|FPRO;

## ESTADOCOM

SINTAXIS: |ESTADOCOM varalfa, varalfa;

PROPÓSITO: Se especifica COM1 o COM2

En FSalida se devuelve el número de caracteres que hay en espera en el buffer.

De momento no se utiliza.

## ET

SINTAXIS: |ET nometi;

DONDE: nometiq es una variable alfanumérica con que etiquetaremos la operación a considerar.

PROPÓSITO: Determina una etiqueta para una zona de programa a la que, posteriormente, volveremos con la sentencia |VETE.

DEVUELVE: Nada.

VER: MANUAL DE PROCESOS: BIFURCACIONES INCONDICIONALES"  
|VETE

EJEMPLO:

|PROCESO salto;|TIPO 0;

(...)

|ET punto1; || A este punto volveremos siempre que la variable FSalida sea 7.

(...)

|SI FSalida = 7;

|VETE punto1; || Salta a donde se encuentre la etiqueta punto1.

|FINSI;

|PARA i = 1;|SI i < 7;|HACIENDO i = i + 1;

(...)

|SI FSalida = 7;

|VETE punto1;

|FINSI;

(...)

|SIGUE;

|FPRC;

**EXTERNOEXE**

SINTAXIS: EXTERNOEXE alfa1;

PROPÓSITO: Poner en alfa1 el ejecutable de sistema a ejecutar, al acabar el programa después de su ejecución volverá a iniciar tu programa.

Si FSalida < 0 no se atiende a la petición ( no hará nada).

## FABRE

SINTAXIS: |FABRE <variable alfanumérica>;

PROPÓSITO: Abre un fichero directo. En la variable numérica, parámetro de la sentencia, vendrá representada la forma en que se abrirá el fichero y su tratamiento. Con toda la familia de estas sentencias podremos tratar ficheros secuenciales, creándolos, grabando en ellos y cerrándolos después. El formato que sigue la variable alfanumérica es el siguiente:

En las primeras tres posiciones se indica el modo de apertura.

r -> lectura

r+ -> lectura y escritura

w -> escritura (borra el contenido anterior)

w+ -> escritura y lectura

Añadiendo al modo de apertura el carácter "t", estamos indicando que el fichero a tratar es de tipo texto. Si por el contrario añadimos "b", el fichero será de tipo binario.

La cuarta posición ha de ser un espacio en blanco y a partir de ella indicaremos el nombre del fichero a abrir.

DEVUELVE:

FSalida > 0, un número Identificador del Fichero Abierto o IFA ( handle ) que después utilizaremos en el resto de la familia de sentencias de Tratamiento de Ficheros Directos o TFD.

FSalida = -1 si el fichero no se ha abierto.

VER: [|FCIERRA](#) [|FLEE](#) [|FGRABA](#) [|FBORRA](#) |TEMPO

EJEMPLO:

```
|PROCESO leer;|TIPO 0;  
    |FABRE "rt datos";  
    f = FSalida; || Devolverá el IFA correspondiente.  
    |SI FSalida ] 0;  
        varalfa = f + " " + 250;  
    |FLEE varalfa; || Sentencia TFD  
    |FINSI;  
    FSalida = f;  
    |FCIERRA FSalida; || Sentencia TFD  
|FPRC;
```

## **FBORRA**

SINTAXIS: |FBORRA <variable alfanumérica>;

DONDE: <variable alfanumérica> contiene el nombre del fichero a borrar.

PROPÓSITO: Borra físicamente del disco el fichero contenido en la variable alfanumérica. Es una sentencia TFD (Tratamiento de ficheros Directos).

DEVUELVE:

FSalida = 0 Si la operación de borrado se ha ejecutado correctamente.

FSalida = -1 En caso de haberse producido algún error.

VER: |TEMPO |[FABRE](#) |[FCIERRA](#)

EJEMPLO:

```
|PROCESO borra;|TIPO 0;  
    |FBORRA "datos"; || Borra el fichero DATOS.  
|FPRC;
```

## FCIERRA

SINTAXIS: |FCIERRA <variable alfanumérica>;

DONDE: <variable alfanumérica> contiene el IFA del fichero abierto.

PROPÓSITO: Cierra el fichero directo cuyo IFA viene indicado en variable alfanumérica. Es una sentencia TFD (Tratamiento de Ficheros Directos).

DEVUELVE:

FSalida = 0 Si la operación de cierre ha sido correcta.

FSalida = -1 En caso de producirse algún error.

VER: |[FABRE](#) |[FLEE](#) |[FGRABA](#)

EJEMPLO:

```
|PROCESO leer;|TIPO 0;  
    |FABRE "rt datos"; || Sentencia TFD  
    f = FSalida; || Obtención del IFA  
    |SI FSalida ] 0;  
        varalfa = f + " " + 250;  
    |FLEE varalfa; || Sentencia TFD  
    |FINSI;  
    FSalida = f;  
    |FCIERRA FSalida; || Cierra el fichero abierto cuyo IFA= f.  
|FPRC;
```

## FGRABA

SINTAXIS: |FGRABA <variable alfanumérica>;

DONDE: <variable alfanumérica> contiene las directrices para grabar información en el fichero.

PROPÓSITO: Graba un texto en el fichero abierto con el siguiente formato:

1a. Posición: IFA de fichero abierto.



2a. Posición: espacio en blanco

Resto de posiciones: texto a grabar.

DEVUELVE:

FSalida > 0 Si la operación de grabación ha sido correcto devolverá el número de caracteres grabados.

FSalida = -1 Si la grabación no se llevado con éxito.

VER: [|FABRE](#) [|FCIERRA](#) [|FLEE](#)

EJEMPLO:

[|PROCESO](#) graba;|TIPO 0;

[|FABRE](#) "wt datos"; || Sentencia TFD

f = FSalida; || Obtención del IFA

[|SI](#) FSalida ] 0;

|| En VARDATOS se guarda el texto a grabar en el  
fichero.

varalfa = f + " " + vardatos;

[|FGRABA](#) varalfa; || Procede con la grabación.

[|FINSI](#);

FSalida = f;

[|FCIERRA](#) FSalida; || Sentencia TFD

[|FPRC](#);

## FINCOM

SINTAXIS: [|FINCOM](#) varalfa;

PROPÓSITO: Cerrar el puerto COM1 o COM2.

Se especifica COM1 o COM2.

**NOTA:** Es muy importante hacer el FINCOM pues si se sale del programa con un COM abierto lo mas posible es que se 'cuelgue'

## FINDIM

SINTAXIS: |FINDIM varnum;

DONDE: varnum variable numérica que representa el buffer previamente dimensionado con [|DIMENSIONA](#).

PROPÓSITO: Una vez hemos dejado de trabajar con el buffer dimensionado liberaremos la memoria que ha ocupado con esta sentencia de manera que se podrá re-usar posteriormente para otro fin.

Es importante saber que al ser los buffers GLOBALES, será IMPRESCINDIBLE liberarlos dentro de la aplicación una vez dejen de utilizarse. La forma burda de liberarlos es saliendo del entorno al sistema operativo pero no es conveniente.

Teniendo en cuenta que tratamos con buffer GLOBALES podríamos hacer uso de esta característica para transportar información masiva entre ficheros ejecutables sin tener que preparar las suficientes variables externas, tan sólo las que contengan el número representativo de cada buffer.

DEVUELVE: Nada.

VER: [|DIMENSIONA](#) [|AL\\_BUF](#) [|DEL\\_BUF](#)

EJEMPLO:

```
|FICHEROS;  
    prueba #0;  
|FIN;  
|VARIABLES;  
    i = 0;  
    buffer = 0;  
    tmp = "";  
|FIN;  
|PROCESO albuffer;|TIPO 0;
```

```

|DIMENSIONA 233,bufter,0;

|SI bufter ] 0

    |AL BUF bufter,0,"";

    |PARA i = 0;|SI i < 233;|HACIENDO i = i + 1;

        tmp = #0i;

        |AL BUF bufter,i,tmp;

    |SIGUE;

    || instrucciones ....

    |FINDIM bufter;

|FINSI;

|FPRC;

```

## FINIMP

SINTAXIS: |FINIMP;

PROPÓSITO: Cerrar la impresora abierta con |IMPRE.

DEVUELVE: Nada

VER: |IMPRE |INFOR |FININF

EJEMPLO:

```

|PROCESO imprimir;|TIPO 0;

    |IMPRE 0;

    |SI FSalida ] 0;

        |INFOR informe;

        |SI FSalida ] 0;

            |BUCLE nombucle;

        |FINSI;

    |FININF;

```

[|FINSI;](#)

[|FINIMP;](#)

|FPRC;

## FININF

SINTAXIS: |FININF;

PROPÓSITO: Descarga el informe cargado con [|INFOR](#) pudiéndose así cargar otro informe distinto.

DEVUELVE:

FSalida = 0 Si la operación se ha desarrollado satisfactoriamente.

FSalida = -1 En caso de error.

VER: [|INFOR](#) [|PRINT](#) [|PIEINF](#)

EJEMPLO:

[|PROCESO](#) imprimir;|TIPO 0;

[|IMPRE](#) 0;

[|SI](#) FSalida ] 0;

[|INFOR](#) informe;

[|SI](#) FSalida ] 0;

[|BUCLE](#) nombucle;

[|FINSI;](#)

[|FININF;](#)

[|FINSI;](#)

[|FINIMP;](#)

|FPRC;

## FINSI

SINTAXIS: |FINSI;

VER: MANUAL DE PROCESOS: BIFURCACIONES CONDICIONALES |SI  
|SINO

EJEMPLO:

|PROCESO imprimir;|TIPO 0;

|IMPRE 0;

|SI FSalida ] 0; || Bifurcación condicional.

|INFOR informe;

|SI FSalida ] 0;|| Bifurcación condicional.

|BUCLE nombucle;

|FINSI;

|FININF;

|FINSI;

|FINIMP;

|FPRC;

## **FINTIMER**

SINTAXIS: |FINTIMER;

PROPÓSITO: Detiene el TIMER activado anteriormente, muy importante el desactivarlo.

## **FINVENTANA**

SINTAXIS: |FINVENTANA nVentana;

PROPÓSITO: Destruye la ventana, donde nVentana es el Valor de FSalida generado con la |VENTANA

## **FLLEE**

SINTAXIS: |FLEE <variable alfanumérica>;

DONDE: <variable alfanumérica> posee un doble papel: contiene la parametrización para la lectura y, tras la lectura, contendrá el texto leído.

PROPÓSITO: Lee un registro o una línea del fichero directo. Dicha lectura está sujeta al siguiente formato:

1a Posición: IFA del fichero abierto.

2a Posición: un espacio en blanco

Resto de posiciones: número máximo de caracteres a leer (el máximo permitido es 250) y para ficheros de texto el número máximo siempre tiene que ser superior a la longitud total de una línea.

DEVUELVE:

FSalida > 0 al ser correcta la operación de lectura, contendrá el número de caracteres leídos.

FSalida = -1 Si por algún motivo la lectura ha sido incorrecta.

VER: |[FABRE](#) |FCIERRA |[FGRABA](#)

EJEMPLO:

|[PROCESO](#) leer;|TIPO 0;

|FABRE "rt datos"; || Sentencia TFD

f = FSalida;

|[SI](#) FSalida ] 0;

|| Procedemos a leer 250 caracteres del fichero abierto con el IFA = f.

varalfa = f + " " + 250;

|[FLEE](#) varalfa; || En varalfa devuelve el texto leído.

|[FINSI](#);

FSalida = f;

|[FCIERRA](#) FSalida; || Sentencia TFD

|FPRC;

## FNOM

SINTAXIS: |FNOM #nvaralfa;

DONDE:

n es el número de fichero.

varalfa es una variable alfanumérica.

PROPÓSITO: Nos permite obtener el nombre del fichero de datos, recogido en varalfa, del fichero n.

DEVUELVE: En varalfa devuelve el nombre del fichero de datos.

EJEMPLO:

Estando en una aplicación cuyo directorio base sea "/u/ds" y la aplicación se llamase "gest".

```
|FICHEROS;  
    clientes #0;  
  
|FIN;  
  
|VARIABLES;  
    vari = "";  
  
|FIN;  
  
|PROCESO prueba;|TIPO 0;  
    || instrucciones ...  
    |FNOM vari;  
    |PINTA 1020, vari;  
    || en vari tendríamos "/u/ds/gest/fich/01/clientes"  
    || instrucciones ...  
  
|FPRC;
```

## F\_TEXTO

SINTAXIS: |F\_TEXTO vn1,valfa1,vn2,vn3,vn4,vn5,vn6,vn7,vn8,vn9,valfa2;

DONDE:

vn1 es una variable numérica que indica el modo con que se va a tratar el fichero de texto:

0 = entrar.

1 = pintar.

2 = pintar y entrar.

3 = presenta la primera hoja.

4 = leer.

5 = grabar.

6 = borrar.

valfa1 es una variable alfanumérica que contendrá el path del fichero de textos.

vn2 es una variable numérica donde se indicará la posición del texto a editar dentro del fichero de textos. Si vn2=0 la sentencia creará un nuevo texto y devolverá en vn2 el número que se le ha asignado.

vn3 y vn4 son variables numéricas que delimitarán el cuadro en pantalla donde aparecerá el texto.

vn5 es una variable numérica que podrá tomar los valores 0 ó 1, dependiendo de si se desea repintar la pantalla o no.

vn6 y vn7 son variables numéricas que marcan el rango de número de líneas a editar.

vn8 es una variable numérica donde se indicará la longitud que tendrá cada línea del texto a editar. El máximo es de 256 caracteres.

vn9 es una variable numérica. Se le asignará el número que representa al buffer donde se cargarán las vn7-vn6 líneas de texto.

valfa2 es una variable alfanumérica que contendrá el título con que aparecerá el texto.



PROPÓSITO: Es un gestor del fichero de textos compactados. Para todos los pequeños textos que se deseen editar sólo existirá un fichero y el acceso a los mismos, dependiendo del lugar donde se produzca la llamada, se realizará a través de vn2 y su edición estará ligada al modo o vn1.

DEVUELVE: Nada.

## GRABA

SINTAXIS: |GRABA abc#nd;

DONDE:

'a' o flag de bloqueo: no usado, siempre a 0.

'b' o flag de mensajes de error:

0 -> Sólo hay mensajes de error en caso de bloqueo de que el registro accedido esté bloqueado por otro puesto o sesión.

1 -> Muestra errores fatales.

2 -> Muestra todos los mensajes.

4 -> No muestra ningún mensaje.

'c' o flag de reintento:

0 -> No reintenta en caso de error.

1 -> Reintenta sólo en caso de bloqueo.

2 -> Reintenta siempre (con cualquier error).

'n' el número de fichero (o el nombre usando la sintaxis *nombre*.)

'd' modo de grabación:

a -> graba registro actual.

n -> graba una clave nueva.

c -> graba un registro nuevo y lo coloca como registro corriente.

b -> igual que c pero además bloquea el registro.

PROPÓSITO: Graba una ficha según los flags indicados.

DEVUELVE:

FSalida = 0 La grabación ha sido correcta.

FSalida > 0 Se ha producido algún error, de indexado o del sistema al grabar.

VER: [|LEE](#) [|ABRE](#) [|CIERRA](#)

EJEMPLO:

```
||PROCESO suma;|TIPO 0;  
  
||ABRE #0;  
  
||LEE 110#0=;  
  
||SI FSalida = 0;  
  
#0#2 = #0#2 + importe;  
  
||GRABA 020#0a;  
  
||FINSI;  
  
||LIBERA #0;  
  
||CIERRA #0;  
  
|FPRC;
```

## GRABABMPPANTALLA

|GRABABMPPANTALLA varalfa1;

PROPÓSITO: Sólo para windows (o cliente windows) graba como bmp la pantalla actual (de la aplicación) en el fichero indicado en varalfa. Útil para hacer manuales.

## GRABADEFSPANTALLAS

|GRABADEFSPANTALLAS varalfa1,varalfa2,varalfa3;

PROPÓSITO: Sólo para windows (o cliente windows) graba como bmp las pantallas indicadas en relación en el fichero especificado en varalfa2 (aquí se indican una lista de defs) en el directorio indicado en varalfa3 pintando las pantallas con la versión actual de runtime y el nombre de empresa especificado en varalfa1. (datos por defecto) . Útil para hacer manuales.

## GRABAMENUPANTALLAS

|GRABAMENUPANTALLAS varalfa1,varalfa2,varalfa3;

PROPÓSITO: Sólo para windows (o cliente windows) graba como bmp las pantallas del menú especificado en varalfa1 presentando las opciones indicadas en la relación contenida en un fichero especificado en varalfa2 (con alfanuméricos: p.ej BAC) en el directorio especificado en varalfa3. Útil para hacer manuales.

## GRABA\_NUMERO

SINTAXIS:|GRABA\_NUMERO alfa, num;

PROPÓSITO: En alfa el canal más un número para indicar el tipo de número (num) a grabar en binario en el fichero.

Tipos de numero:

1 = short (2 bytes)

2 = entero (4 bytes)

3 = float (4 bytes)

4 = double (8 bytes)

## GRABACOM

SINTAXIS: |GRABACOM varalfa1,varalfa2;

PROPÓSITO: Envía datos al puerto.

DONDE:

En varalfa1: Se especifica COM1 o COM2,

En varalfa2: String de datos a grabar.

## GRABAENDEF

SINTAXIS: |GRABAENDEF valfa1, valfa2[{vnum1,vcm1}[, {...}][, ...]]

DONDE:

valfa1 es una variable alfanumérica donde se indicará el path completo del fichero definidor que deseamos crear o modificar.

valfa2 es una variable alfanumérica donde se indicará el path completo del fichero de datos donde se grabarán los datos correspondientes.

vnum1 variable numérica que representa el número de campo a modificar.Opcional.

vcm1 variable o campo. Recoge el valor modificado.

PROPÓSITO: Esta sentencia permite crear si no existe, o modificar si existe, un fichero de datos a través de su def sin que deba estar declarado en la declaración de ficheros. Este acceso, paralelo a la declaración previa de ficheros, se lleva a cabo indicando en valfa1 el path del fichero definidor, donde se encuentra la estructura de campos, en valfa2 el path del fichero de datos y como parámetros opcionales, indicaremos, en grupos de dos elementos, el número de campo del definidor y una variable o campo que contendrá la información a grabar en el fichero de datos.

Se pueden indicar tantas parejas de valores como se quieran, dependiendo de los campos a tratar.

Esta sentencia es muy práctica si se quiere actualizar un número reducido de campos de un fichero de datos sin tener que cargar su fichero definidor.

DEVUELVE: Nada

VER: |LEEENDEF

EJEMPLO:

```
|PROCESO graba_endef;  
    |DEFICO dfich;  
    origen = dfich + "/01/pepe";  
    destino = dfich + "/02/pepe";  
    |GRABAENDEF origen, destino, 0,#1#0,1,#1#1,2,#1#2;  
|FPRC;
```

**GRABASECUENCIAL**

SINTAXIS: |GRABASECUENCIAL varalfa, vnum1, vnum2, vnum3;

DONDE:

varalfa es una variable alfanumérica que contiene el path completo del fichero secuencial a grabar.

vnum1 es una variable numérica y su valor representa el buffer donde se encuentra la información a grabar.

vnum2 es una variable numérica y representa la posición inicial del buffer a partir de la cual hay que grabar.

vnum3 es una variable numérica y representa la posición final del buffer que queremos grabar.

PROPÓSITO: Graba la información, de carácter alfanumérico, contenida en el buffer representado por vnum1 en un fichero secuencial con el path varalfa. Utilizando los parámetros vnum2 y vnum3 podemos delimitar la información transferida.

DEVUELVE:

FSalida > 0 El número de líneas grabadas.

FSalida = -1 en caso de haberse producido algún error.

VER: [|LEESECUENCIAL](#)

EJEMPLO:

```
|FICHEROS;  
    terminal #0;  
  
|FIN;  
  
|VARIABLES;  
    i = 0;  
    buffer = 0;  
    tmp = "";  
    defecto = "/Agi/datos/terminal.ter";  
  
|FIN;
```

```

|PROGRAMA;

    || instrucciones ...

    |DIMENSIONA 233,bufter,0;

    |AL\_BUF bufte,0,"";

    |PARA i = 0;|SI i < 233;|HACIENDO i = i + 1;

    tmp = #0i;

    |AL\_BUF bufte,i,tmp;

    |SIGUE;

    |GRABASECUENCIAL defecto,bufte,0,233;

    |FINDIM bufte;

    || instrucciones ...

|FPRO;

```

## GRABAVE

SINTAXIS: |GRABAVE p,q,nombre;

DONDE: p es la posición inicial de la pantalla, q la posición final, obtenidas a partir de la operación (fila\*100+columna), y nombre el nombre del fichero donde vamos a grabar la ventana.

PROPÓSITO: Grabar en disco una zona delimitada de la pantalla denominada también ventana.

DEVUELVE:

FSalida = 0 Si la grabación ha sido correcta.

FSalida = -1 En caso de error.

VER: |TEMPO |[PINVE](#) |[FBORRA](#)

EJEMPLO:

```

|PROCESO grabave;|TIPO 0;

    || Graba una pantalla sin peligro de

```

```
|| que otro usuario la 'machaque'  
|TEMPO fichero;  
|GRABAVE 1010,2070,fichero;  
...  
|FBORRA fichero; || Importante  
|FPRC;
```

## GRAF

SINTAXIS: |GRAF número;

DONDE:

número puede tomar los siguientes valores:

- 1 -> activa modo gráfico en el terminal.
- 2 -> desactiva modo gráfico en el terminal.
- 0 -> pinta raya vertical.
- 1 -> pinta raya horizontal.
- 2 -> pinta esquina superior izquierda.
- 3 -> pinta cruce superior.
- 4 -> pinta esquina superior derecha.
- 5 -> pinta cruce izquierdo.
- 6 -> pinta cruce central.
- 7 -> pinta cruce derecha.
- 8 -> pinta esquina inferior izquierda.
- 9 -> pinta cruce inferior.
- 10 -> pinta esquina inferior derecha.
- 11 -> pinta gráfico 1.
- 12 -> pinta cuadro gráfico 2.

PROPÓSITO: Pintar el gráfico correspondiente a número.

DEVUELVE: Nada.

VER: [|PINTA](#) [|ATRI](#) [|CUADRO](#) [|CUADRO\\_D](#) [|CUADRO\\_S](#)

EJEMPLO:

```
||PROCESO pintar;|TIPO 0;

    || pintar un cuadro

    ||GRAF -1; || Activa el modo gráfico.

    ||PINTA 1010; || Pintar esquinas

    ||GRAF 2;

    ||PINTA 1070;

    ||GRAF 4;

    ||PINTA 2010;

    ||GRAF 8;

    ||PINTA 2070;

    ||GRAF 10;

    || raya superior

    ||PARA i = 1011;||SI i < 1070;||HACIENDO i=i+1;

        ||PINTA i;

        ||GRAF 1;

    ||SIGUE;

    || raya inferior

    ||PARA i = 2011;||SI i < 2070;||HACIENDO i=i+1;

        ||PINTA i;

        ||GRAF 1;

    ||SIGUE;

    || raya vertical izquierda
```



|[PARA](#) i =1110;|[SI](#) i < 2010;|[HACIENDO](#) i=i+100;

|[PINTA](#) i;

|[GRAF](#) 0;

|[SIGUE](#);

|| raya vertical derecha

|[PARA](#) i =1170;|[SI](#) i < 2070;|[HACIENDO](#) i=i+100;

|[PINTA](#) i;

|[GRAF](#) 0;

|[SIGUE](#);

|[GRAF](#) -2; || Desactiva modo gráfico.

|FPRC;

Este ha sido un ejemplo de cómo podríamos implementar un cuadro gráfico, aunque resulta más práctico utilizar la sentencia [CUADRO](#) . La sentencia [PINTA](#) posiciona el cursor donde deseamos y la sentencia [GRAF](#) se encarga de pintar el carácter gráfico seleccionado.

## HACIENDO

SINTAXIS: |[HACIENDO](#) [operación];

VER: MANUAL DE PROCESOS: BUCLES DE INSTRUCCIONES |[PARA](#) |[SI](#) |[SIGUE](#)

EJEMPLO:

|[PROCESO](#) raya;|TIPO 0;

|[GRAF](#) -1;

|[PARA](#) i =1170;|[SI](#) i < 2070;|[HACIENDO](#) i=i+100;

|[PINTA](#) i;

|[GRAF](#) 0;

|[SIGUE](#);

|[GRAF](#) -2;

|FPRC;

## HAZ

SINTAXIS: |HAZ nomcalc;

DONDE: nomproc es el proceso que se ejecutará desde esta línea de programa provocando una bifurcación del flujo del proceso general.

PROPÓSITO: Llama a otro proceso, definido en el mismo fichero.

DEVUELVE: Nada.

VER: MANUAL DE PROCESOS: ANIDACION.

EJEMPLO:

|[PROCESO](#) sumar;|TIPO 0;

total = total + 1;

|FPRC;

|[PROCESO](#) inicial;|TIPO 0;

|[HAZ](#) suma;

|FPRC;

## HORA

SINTAXIS: |HORA varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Devuelve la hora del sistema en formato HH:MM:SS

EJEMPLO:

|[PROCESO](#) dahora;|TIPO 0;

|[HORA](#) vhora;

|[PINTA](#) 1020, vhora;

|FPRC;

## INICIOCOM

SINTAXIS: |INICIOCOM varalfa1, varnumérica1, varalfa2, varnumérica2, varnumérica3, varnumérica4;

PROPÓSITO: Instrucción bajo MSDOS para el manejo del COM1 y COM2 usando las interrupciones internas, de modo que no se pierden caracteres al usar un buffer.

DONDE:

Varalfa1: Si es COM1 o COM2

Varnumérica1: velocidad en baudios ( de momento por razones de conversión numérica limitado a un máximo de 19200)

Varalfa2: Paridad (N=nonen E=even)

Varnumérica2: Bits (Normalmente 8 o 7

Varnumérica3: Stop bits (1 o 2)

Varnumérica4: Tamaño del buffer (máximo 32000 bytes)

VER: |INICIOCOM |[LEECOM](#) |[GRABACOM](#) |[ESTADOCOM](#) |[LEECOMBUFFER](#)

## INCLUYE

SINTAXIS: |INCLUYE nomfichero;

PROPÓSITO: Esto es una especie de declaración pues va fuera de los procesos. Cuando se encuentra un INCLUYE, digamos se sustituye la instrucción por todo el contenido del fichero, si no se pone path se busca en el directorio standard de procesos, si se pone :/path el :/ se sustituye por el directorio base, y si se pone un path se lee el nomfichero en el path indicado.

DEVUELVE: Nada.

## IMPRE

SINTAXIS: |IMPRE n;

DONDE:

n el número de impresora definido en la empresa.

Si n=-77 (a partir del básico 9.09V-2), y en la variable Impresora hemos definido una ruta y un nombre de archivo, direcciona la impresión a ese archivo de spool.

P.Ej:

Impresora = "/u/ds/sociedad/tmp/fichero.txt";

IMPRE -77

PROPÓSITO: Selecciona la impresora n según el número de sentencia que le correspondería al campo definido como impresora 'I' en el def de empresa, siendo 0 la primera y, en caso de no haber ninguna definida, se toma por defecto ibm80.

DEVUELVE:

FSalida >= 0 Si la apertura de la impresora seleccionada ha sido correcta.

FSalida = -1 En caso de no existir la impresora seleccionada o de no encontrarse en condiciones.

VER: |[FINIMP](#) |[INFOR](#)

EJEMPLO:

|[PROCESO](#) imprimir;|TIPO 0;

|[IMPRE](#) 1; || por defecto impresora 1

|[SI](#) FSalida ] 0;

|[INFOR](#) informe;

|[SI](#) FSalida ] 0;

|BUCLE nombucle;

|[FINSI](#);

|[FININF](#);

[|FINSI;](#)

[|FINIMP;](#)

|FPRC;

## IMPRIME

SINTAXIS: |IMPRIME varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Direcciona la cadena alfanumérica varalfa a la impresora activada.

DEVUELVE: FSalida -1 error al escribir en impresora.

VER: [|IMPRE](#)

EJEMPLO:

[|PROCESO](#) imprimir;|TIPO 0;

[|IMPRE](#) 1; || por defecto impresora 1

[|SI](#) FSalida ] 0;

[|IMPRIME](#) "CODIGO DE PRODUCTO";

|| Imprimiría "CÓDIGO DE PRODUCTO" en la impresora.

[|FINSI;](#)

[|FINIMP;](#)

|FPRC;

## INFOR

SINTAXIS: |INFOR <variable alfanumérica>;

PROPÓSITO: Carga el informe definido en la variable alfanumérica que puede incluir el path completo. Si el nombre no tiene directorio o no se indica, por defecto, se buscará en el directorio de ficheros definidores o 'def'. No hay que ponerle extensión, ya que se le añade automáticamente la extensión '.inf'.

Únicamente puede trabajarse con un informe a la vez, es decir no puede ser cargado un segundo informe mientras el primero no haya finalizado.

DEVUELVE:

FSalida = 0 Si el informe ha sido encontrado y cargado con normalidad.

FSalida = -1 Si se ha producido algún error.

VER: [|FININF](#) [|PRINT](#) [|PIEINF](#) [|IMPRE](#)

EJEMPLO:

```
|VARIABLES;  
    informe = "factura";  
|FIN;  
  
|PROCESO imprimir;|TIPO 0;  
  
    |IMPRE 0;  
  
    |SI FSalida ] 0;  
  
        |INFOR informe; || Carga el informe FACTURA  
  
        |SI FSalida ] 0;  
  
            |BUCLE nombucle;  
  
        |FINSI;  
  
        |FININF;  
  
    |FINSI;  
  
    |FINIMP;  
  
|FPRC;
```

## IP\_REMOTO

SINTAXIS: [|IP\\_REMOTO](#) alfa;

DEVUELVE: En alfa se cargará en formato texto la *ip* del cliente o puesto, si no es cliente servidor (local) vuelve vacía.

## LEE

SINTAXIS: |LEE abc#nd;

DONDE:

'a' o el flag de bloqueo puede ser:

0 -> sin bloqueo

1 -> con bloqueo

'b' o el flag de mensajes de error puede ser:

0 -> sólo hay mensajes de error en caso de bloqueo.

1 -> errores fatales

2 -> todos los mensajes

4 -> no da mensajes

'c' o flag de reintento puede ser:

0 -> no reintenta en caso de error

1 -> reintenta sólo en caso de bloqueo

2 -> reintenta siempre (con cualquier error)

'n' será el número de fichero que tengamos definido en la parte |FICHEROS o su nombre.

'd' será el modo de lectura y puede ser:

c -> lee el registro corriente

s -> lee el siguiente

a -> lee el anterior

p -> lee el primero

u -> lee el último

= -> lee el registro igual a la clave

> -> lee el registro mayor que la clave

] -> lee el registro mayor o igual que la clave

PROPÓSITO: Lee un registro de un fichero según parametrización elegida.

DEVUELVE:

FSalida = 0 si la lectura ha sido correcta.

FSalida > 0 si se ha producido algún error de indexado o del sistema operativo.

FSalida < 0 si la lectura no ha encontrado lo pretendido.

VER: |[ABRE](#) |[CIERRA](#) |LIBERA |[GRABA](#) |[BORRA](#)

EJEMPLO:

```
|PROCESO suma;|TIPO 0;

|ABRE #0;

|LEE 110#0=;      || Procede a leer el fichero 0 con
bloqueo,

                        || mostrando los errores fatales solamente,
                        || no reintentando en caso de error y leerá
                        || el registro igual a la clave de búsqueda.

|SI FSalida = 0;

        #0#2 = #0#2 + importe;

|GRABA 020#0a;

|FINSI;

|LIBERA #0;

|CIERRA #0;

|FPRC;
```

## LEECOM

SINTAXIS: |LEECOM varalfa,varnumérica,varalfa;

PROPÓSITO: Se especifica COM1 o COM2,

Máximo de caracteres a leer,



## Destino de la lectura

**NOTA:** Ojo, esta rutina lee y vacía el buffer pero no se espera a que éste se llene de modo que si no hay nada en el buffer no pone nada en la variable de destino, en FSalida se devuelve el número de caracteres recogidos.

### LEECOMBUFFER

SINTAXIS: |LEECOMBUFFER varalfa, varnumerica, varalfa;

PROPÓSITO: Igual que [LEECOM](#) pero sin vaciar buffer. Sirve para saber lo que hay en el buffer sin tocarlo.

### LEENDEF

SINTAXIS: |LEENDEF valfa1, valfa2[, {vnum1,vcm1}[, {...}[, ...]]]

DONDE:

valfa1 es una variable alfanumérica donde se indicará el path completo del fichero *ata* que deseamos leer.

valfa2 es una variable alfanumérica donde se indicará el path completo del fichero de datos donde se leerán los datos correspondientes.

vnum1 variable numérica que representa el número de campo a leer.  
Opcional.

vcm1 variable o campo. Recoge el valor leído.

PROPÓSITO: Leer determinados campos de un fichero de datos configurado según su correspondiente *ata* sin tener que haberlo declarado en la DECLARACIÓN DE FICHEROS. En algunos casos en los que la consulta de un fichero de datos específico sea reducida, no interesa declararlo forzando a que se cargue su fichero con lo que utilizaremos esta sentencia para obtener información muy particular del fichero de datos de forma sencilla y rápida. El fichero se localizará según el path dado por valfa1 y el path del fichero de datos lo indicaremos en valfa2. La información a leer se especificará en los parámetros opcionales siguientes que deberán incluirse en grupos de dos: el número de campo o vnum1 y una variable o campo, vcm1, donde recogeremos el valor leído.

En caso de no incluir ningún parámetro opcional, la sentencia leerá el registro por defecto creando el fichero en caso de que no exista.

DEVUELVE: Nada

VER: [GRABAENDEF](#)

## LEE\_NUMERO

SINTAXIS: |LEE\_NUMERO alfa, num;

PROPÓSITO: Escribir en alfa el canal más un numero para indicar el tipo de numero (num) a leer en binario del fichero.

## LEESECUENCIAL

SINTAXIS: |LEESECUENCIAL varalfa, vnum1, vnum2, vnum3, vnum4;

DONDE:

varalfa: variable alfanumérica que contiene el path del fichero secuencial a leer.

vnum1: variable numérica que representa el buffer dimensionado.

vnum2: variable numérica cuyo valor equivale al número de líneas del fichero secuencial leído.

vnum3: variable numérica. Representa el número de líneas anteriores a las leídas del fichero secuencial.

vnum4: variable numérica. Representa el número de líneas posteriores a las leídas del fichero secuencial.

PROPÓSITO: Lee un número vnum2 de líneas de un fichero secuencial ubicado según el path dado en varalfa creando un buffer vnum1 y dimensionándolo según vnum2 y los valores indicados en vnum3 y vnum4. Estas dos últimas cantidades equivalen a un número determinado de líneas en blanco, pero del mismo tipo que las leídas del secuencial, que podemos incluir tanto al principio, vnum3, como al final, vnum4, del bloque de vnum2 líneas. Por tanto el buffer será sobredimensionado y su tamaño real, el número de

líneas, será devuelto en la variable vnum2 y su valor será igual a la suma de las cantidades vnum2, vnum3, vnum4.

DEVUELVE:

vnum1 > 0 el número que representa al buffer donde se ubicará la información leída del fichero secuencial.

vnum1 < 0 En caso de producirse algún error al dimensionar el buffer o leerlo.

vnum2 > 0 Número total de líneas leídas del fichero secuencial.

VER: [|DIMENSIONA](#) |GRBASECUENCIAL

EJEMPLO:

|FICHEROS;

terminal #0;

|FIN;

|VARIABLES;

i = 0;

bufter = 0;

totter = 0;

tmp = "";

defecto = "/Agi/datos/terminal.ter";

|FIN;

|PROGRAMA;

|| instrucciones ...

[|LEESECUENCIAL](#) defecto,bufter,totter,0,0;

[|SI](#) FSalida ] 0;

[|PARA](#) i = 1;[|SI](#) i < totter; [|Y](#) i < 233;[|HACIENDO](#) i = i

+ 1;

[|DEL\\_BUF](#) bufter,i,tmp;

```

                                #0i = tmp;

                                |SIGUE;

                                |FINDIM buffer;

                                |FINI;

                                || instrucciones ...

                                |FPRO;

```

## LEETECLA

SINTAXIS: |LEETECLA varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Espera la pulsación de una tecla.

DEVUELVE:

Teclas normales: varalfa devuelve la letra correspondiente.

Teclas especiales: varalfa devuelve su código interno en formato 800:  
802 = Return.

Para ver los códigos de cada tecla entrar en UTILIDADES DIAGRAM 2.0 → Configuración de terminales → Generar → A partir de la cuarta pantalla están todos los códigos internos de las teclas.

EJEMPLO:

```

                                |FICHEROS;

                                datos #0;

                                |FIN;

                                |VARIABLES;

                                num_tec = "";

                                |FIN;

                                |PROGRAMA;

                                num_tec = " ";

                                |PARA ;|SI num_tec < 1;|O num_tec > 5; |HACIENDO;

```

```
|LEETECLA num_tec;  
|SI num_tec ] 1;|Y num_tec [ 5;  
    || instrucciones ...  
|FINSI;  
    || instrucciones ...  
|SIGUE;  
|FPRC;
```

## LETRA

SINTAXIS: |LETRA var;

DONDE: var es una variable alfanumérica que posee un doble papel: por una parte contendrá el número a convertir y por otra, tras la conversión, contendrá una cadena de caracteres.

PROPÓSITO: Escribe el número introducido en letras (en castellano).

DEVUELVE: Nada.

EJEMPLO:

```
|PROCESO numero;|TIPO 0;  
    alfa = 123;  
    |LETRA alfa;  
    |PINTA 1001,alfa; || pinta CIENTO VEINTITRÉS  
|FPRC;
```

## LIBERA

SINTAXIS: |LIBERA #n;

DONDE: n el número de fichero a liberar.

PROPÓSITO: Desbloquear todos los registros del fichero n bloqueados al leerlos. Se recomienda bloquear un solo registro a la vez por fichero.

DEVUELVE:

FSalida = 0 El desbloqueo de TODO el fichero ha sido correcta.

FSalida > 0 Caso de error de indexado o del sistema.

VER: [|LEE](#) [|ABRE](#) [|CIERRA](#) [|GRABA](#) [|BORRA](#) [Normas para el bloqueo de registros](#)

EJEMPLO:

[|PROCESO](#) suma;|TIPO 0;

[|ABRE](#) #0;

[|LEE](#) 110#0=;

[|SI](#) FSalida = 0;

#0#2 = #0#2 + importe;

[|GRABA](#) 020#0a;

[|FINSI](#);

[|LIBERA](#) #0; || Libera el fichero 0 porque se ha leído bloqueando los registros.

[|CIERRA](#) #0;

|FPRC;

Recomendamos LIBERAR antes de leer ya que de esta manera sabremos siempre que esta bloqueado el registro al grabar y al realizar otra lectura desbloquearía el anterior.

## LINEAINFORME

SINTAXIS: |LINEAINFORME varnum1,varnum2;

Si varnum1 = 0 en varnum2 viene el número de línea corriente en la impresión del informe actual.

Si varnum1 = 1 se pone como línea corriente de impresión el valor de varnum2.

## LINEAL\_SIMPLE

SINTAXIS: |LINEAL\_SIMPLE #fichero#clave, flag, pi, pf, proceso\_lim1, proceso\_lim2, proceso\_evento;

DONDE:

- flag: numérica
  - 1 = Si se suma es mantenimiento (sino es consulta).
  - 2 = Si se suma se permite el cambio de clave (orden).
  - 4 = Si se suma la ventana está enmarcada en las coordenadas indicadas según la pantalla, sino es independiente.
  - 8 = Si se suma la ventana NO es reposicionable (si no tiene caption).
  - 16 = Si se suma la ventana tiene borde sencillo sino borde doble.
  - 32 = Si se suma es un control modeless (independiente) En FSalida devuelve el código del control.
  - 64 = Ordenado por número de campo en vez de orden de pantalla.
  - 128 = Rango de clave no simple (como en el entlieal).
- pi, pf (numéricas) : marco (si aplicable) del grid.
- proceso\_lim1, proceso\_lim2:

Procesos ( o NULL) indicando el limite inferior y superior del lineal, ojo es una limitación simple y en caso de claves compuestas el resultado es como si todos los componentes de la clave fueran uno solo.
- proceso\_evento:

Si no es NULL se ejecuta este proceso para los siguientes eventos indicados por el valor de FSalida al ejecutarlo:

  - 1 – Nuevo corriente
  - 2 – Al entrar al lineal

3 – Al salir del lineal

4 – Para activar en algún campo del lineal un doble clic. Si al termino de la ejecución le ponemos FSalida = "SKIPDEFAULT" el lineal se quedará como antes.

DEVUELVE: FSalida = El andel del control o –1 en caso de error.

## MANTE

SINTAXIS: |MANTE #n; o |MANTE #Pn;

DONDE: n representa el número de fichero al que hay que realizarle un proceso de mantenimiento.

PROPÓSITO: Realiza el mantenimiento standard con el fichero n. La forma #Pn obliga a no pintar la primera pantalla del fichero inicialmente.

DEVUELVE:

FSalida = 0 si el mantenimiento se ha realizado normalmente.

FSalida = -1 si, por algún motivo, se ha producido algún error.

VER: [|CLS](#) [|CABEZA](#) [|PINPA](#) [|ENDATOS](#)

EJEMPLO:

|FICHEROS;

clientes#0;

|FIN;

|PROGRAMA;

[|CLS](#);

[|CABEZA](#) "ESTE ES EL TITULO";

[|MANTE](#) #0; || Provoca el mantenimiento normal del fichero

de clientes.

|FPRO;



Este proceso también podría haberse escrito de la siguiente forma:

```
|FICHEROS;
```

```
    clientes#0;
```

```
|FIN;
```

```
|PROGRAMA;
```

```
    |CLS;
```

```
    |CABEZA "ESTE ES EL TITULO";
```

```
    |PINPA #0#0;
```

```
    |MANTE #P0;
```

|| Provoca el mantenimiento del fichero de clientes tras pintar su primera pantalla.

```
|FPRO;
```

## **MENAV**

SINTAXIS: |MENAV v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Pinta un mensaje en la pantalla.

DEVUELVE: Nada.

VER: |[ERROR](#) |[AVISO](#) |[PAUSA](#) |[MENSA](#) |[ATRI](#)

EJEMPLO:

```
|PROCESO control2;|TIPO 0;
```

```
    |SI #0Campo = "N";
```

```
        |MENAV "0000No esta permitido 'N'";
```

```
        |ERROR;
```

```
        |ACABA;
```

```
    |FINSI;
```

|FPRC;

## MENSA

SINTAXIS: |MENSA v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Pinta un mensaje en pantalla. Hay que poner un [|PAUSA](#) si queremos que espere tras escribir el mensaje, sino el mensaje se borra solo.

DEVUELVE: Nada.

VER: [|BLIN](#) [|MENAV](#) [|PAUSA](#) [|ATRI](#)

EJEMPLO:

|PROCEDO control1;|TIPO 7;

[|MENSA](#) "0000Entre de todo menos 'N'";

|FPRC;

## MENU

SINTAXIS: |MENU varpunt;

DONDE:

varpunt es una variable puntero y deberá apuntar a una serie de variables alfanuméricas que en sentencia contendrán:

1ª.- Posición del menú en pantalla. Para determinarla consideraremos el resultado de la operación  $\text{fila} * 100 + \text{columna}$ ; si columna = 0 se centra el menú automáticamente.

2ª.- Opción por defecto.

3ª.- Texto que precede al menú.

4ª.- Letras que identifican cada opción y cuyo número determina el número total de opciones del menú.

5ª.- Todas las opciones, tantas como letras en la cuarta variable.

DEVUELVE:

FSalida > 0, el número de opción seleccionado

FSalida = 0, si se ha pulsado escape

FSalida = -1, si ha pulsado CTRL-C

VER: [|MENUG](#)

EJEMPLO:

```
|VARIABLES;  
  
    {-1}menu = "";  
  
    menu1 = "2400";  
  
    menu2 = "1";  
  
    menu3 = "Elija:"  
  
    menu4 = "MTI";  
  
    menu5 = "pantalla Mes";  
  
    menu6 = "pantalla Totales";  
  
    menu7 = "Impresora";  
  
|FIN;  
  
|PROCESO crear_menú;  
  
    |MENU menú;  
  
|FPRC;
```

## **MENUG**

SINTAXIS: [|MENUG](#) varpuntero;

DONDE: varpuntero es una variable puntero con las opciones necesarias.

PROPÓSITO: Pinta un menú de la misma manera que la sentencia [|MENU](#) con la diferencia de utilizar un formato de cuadro. Con respecto a la función [|MENU](#) las entradas de las letras y el texto se omiten y en su lugar se pone otra que contiene el número de opciones.

DEVUELVE:

FSalida > 0, el número de opción seleccionado

FSalida = 0, si se ha pulsado escape

FSalida = -1, si ha pulsado CTRL-C.

VER: [|MENU](#)

EJEMPLO:

```
|VARIABLES;  
  
    {-1}menú = "";  
  
    menu1 = "1400";      || posición en pantalla  
  
    menu2 = "1";        || opción por defecto  
  
    menu3 = "3";        || numero de opciones  
  
    menu3 = "pantalla Mes";  || primera opción  
  
    menu4 = "pantalla Totales"; || segunda opción  
  
    menu5 = "Impresora";    || tercera opción  
  
|FIN;  
  
|PROCESO crear_menú;  
  
    |MENUG menú;  
  
|FPRC;
```

## **MENUIITEM**

SINTAXIS: |SYSTEM <variable numérica>, <variable alfanumérica>.

PROPÓSITO: Ejecuta la opción de un menu.

La variable numérica guardará el menú. Si se le pone 0 es el menú principal.

La variable alfanumérica contendrá tres elementos numéricos separados por comas:

Primero: posición submenú.

Segundo: posición ítem dentro del submenú.

Tercero: acción a seguir:

0 = ejecutar (igual que si tú pincharas en la opción).

1 = obtener estado ( [UN] CHECKED, GRAYED, SEPARATOR ).

2 = obtener nombre de la opción.

3 = obtener el id de la opción (enter otras sirve en 10 para ejecutarla directamente).

DEVUELVE: En FSalida devuelve vacío si no está soportada la orden. Si no, el resultado según la operación.

EJEMPLO:

|MENUITEM 0, "0,0,0"; || submenú 0 opción 0 -> ejecútala

## **MODO\_BMP**

SINTAXIS: |MODO\_BMP num1,num2;

DONDE:

Si num1 = 0; pone en num2 el actual valor de pintado de bmps:

0 = normal

1= encuadra

2 = tal cual

Si num1 = 1 pone el valor de pintado según num2.

En el DS.INI:

ModoBmp = 0 (o 1 o 2)

Pone el modo general de pintado del bmp (por defecto 0).

## **MODO\_RELACION**

SINTAXIS: |MODO\_RELACION #fichero, modo, nombre\_fichero\_relación, estadoalta, estadoobliga;

DONDE:

modo -> numérica

modo = 0 leer

modo = 1 poner

nombre\_fichero\_relacion -> alfa

fichero relacionado (el nombre) que se busca en las relaciones de "#fichero"

estadoalta -> alfa

estadoalta = "S" alta permitida si

estadoalta = "N" alta permitida no

estadoobliga -> alfa

estadoobliga = "S" existencia obligatoria si

estadoobliga = "N" existencia obligatoria no

DEVUELVE:

FSalida = 0 si la relación existe

FSalida = -1 si la relación no existe

## MODULO

SINTAXIS: |MODULO varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Carga en varalfa el nombre del ejecutable (Agi) actual.

EJEMPLO:

Dado un programa llamado clientes.mdr

|MODULO varalfa;

|| varalfa contendrá "clientes".

## MKDIR

SINTAXIS: |MKDIR alfa1;

PROPÓSITO: En alfa1 el directorio a crear, ojo, crear directorios de uno en uno.

Si FSalida < 0 error.

## NOALARMA

SINTAXIS: |NOALARMA;

PROPÓSITO: Desactiva la posibilidad de interrupción de un proceso por la activación de la alarma. Es imprescindible en los procesos en los que se programe la alarma.

DEVUELVE: Nada.

VER: [|ALARMA](#) |SIALARMA

**ATENCIÓN:** Deberíamos tener en cuenta que al entrar al Runtime por primera vez la alarma se activa si no le decimos lo contrario. Por tanto debemos utilizar esta instrucción para desactivar esta posibilidad en algún punto en concreto, pero teniendo CUIDADO en ejecutar después un |SIALARMA para que quede de nuevo activada la alarma, ya que si no, hasta que no se vuelva a entra al Runtime, no se volverá a ejecutar.

EJEMPLO:

```
|FICHEROS;
```

```
    datos #0;
```

```
|FIN;
```

```
|PROGRAMA;
```

```
    |NOALARMA;
```

```
    || mientras se este ejecutando este programa no se activara  
la alarma
```

```
    || instrucciones ...
```

```
    |SIALARMA;
```

|FPR10;

## **NOMAP**

SINTAXIS: |NOMAP v;

DONDE: v es una variable alfanumérica.

PROPÓSITO: Entrega el nombre de la aplicación.

DEVUELVE: Nada.

EJEMPLO:

Si la aplicación esta en el directorio /ds/gest/ en aplicación devolvería gest.

|[NOMAP](#) aplicación;

|| aplicación = "gest".

## **NOMBRE\_IP**

SINTAXIS: |NOMBRE\_IP alfa1,alfa2;

PROPÓSITO: Introduciremos en alfa1 la IP y en alfa2 devolverá el nombre (¡¡si es que el host lo puede resolver!!).

## **NOME\_DAT**

SINTAXIS: |NOME\_DAT #n <variable alfanumérica>;

DONDE: n un número de fichero de datos.

PROPÓSITO: Pone el contenido de la variable alfanumérica como nombre del fichero de datos del fichero n. Tener en cuenta que para un mismo def podemos disponer de varios ficheros de datos, si en un momento dado deseamos dar de alta una ficha en algunos o todos sus ficheros de datos podemos utilizar esta sentencia para ejecutar dicha operación. La variable no debe exceder en más de 8 caracteres y ninguno de ellos debe ser '/' o '\'. Si el fichero está abierto deberá cerrarse primero y volverlo a abrir para que surta efecto.



DEVUELVE: Nada.

VER: |[PATH\\_DAT](#)

EJEMPLO:

```
|FICHEROS;  
    ctcuen1#0;  
  
|FIN;  
  
|PROCESO grabacu;|TIPO 0;  
    |NOME\_DAT #0 "ctcuen1";  
    || Primer fichero de datos del fichero a CTCUEN1.  
  
    |ABRE #0;  
    |GRABA 020#0n;  
    |CIERRA #0;  
    |NOME\_DAT #0 "ctcuen2";  
    || Segundo fichero de datos del fichero a CTCUEN2.  
  
    |ABRE #0;  
    |GRABA 020#0n;  
    |CIERRA #0;  
  
|FPRC;
```

## NOPARA

SINTAXIS: |NOPARA;

PROPÓSITO: Una vez se ejecuta esta sentencia, no se permitirá ninguna interrupción en un bucle declarado, desde teclado.

DEVUELVE: Nada.

VER: |SIPARA

EJEMPLO:

|[PROCESO](#) imprimir;|TIPO 0;

|[IMPRE](#) 0;

|[SI](#) FSalida ] 0;

|INFORME informe;

|[SI](#) FSalida ] 0;

|[NOPARA](#); || No podremos detener la  
impresión del fichero por su bucle declarado.

|[BUCLE](#) nombucle;

|[FINSI](#);

|[FININF](#);

|[FINSI](#);

|[FINIMP](#);

|FPRC;

## ONTIMER

SINTAXIS: |ONTIMER varnumérica, varalfa;

PROPÓSITO: Se especifica cada cuantos 1/18 segundos se activa, con el mismo formato que las teclas de función se especifica la rutina externa o mdr que se debe ejecutar al activarse el timer.

## PARA

SINTAXIS: |PARA [operación];

DEVUELVE: Nada.

VER: [MANUAL DE PROCESOS: BUCLE DE INSTRUCCIONES](#) |[HACIENDO](#)  
|SIGUE

EJEMPLO:

|[PROCESO](#) raya;|TIPO 0;

|[GRAF](#) -1;  
|[PARA](#) i =1170;|[SI](#) i < 2070;|[HACIENDO](#) i=i+100;  
|[PINTA](#) i;  
|[GRAF](#) 0;  
|[SIGUE](#);  
|[GRAF](#) -2;  
|FPRC;

## PARTE\_FICHERO

SINTAXIS: |PARTE\_FICHERO aAlfa1, nNume1, aAlfa2;

DONDE:

aAlfa1: fichero a partir

nNume1: Número de bytes máximos para cada fragmento

aAlfa2: Nombre base de los fragmentos.

Por ejemplo:

|[PARTE\\_FICHERO](#) "prueba.tgz", 1450000, "prueba.";

esto generará "prueba.0", "prueba.1", etc.

## PATH\_DAT

SINTAXIS: |PATH\_DAT #n <variable alfanumérica>;

DONDE: n un número de fichero.

PROPÓSITO: Pone el contenido de una variable alfanumérica como directorio del fichero de datos del fichero n. La variable no debe exceder en más de 42 caracteres y los directorios se deben separar con '/'. No se debe usar '\'. Si el fichero ya ha sido abierto esta sentencia no tiene efecto a menos que se cierre y se vuelva a abrir.

DEVUELVE: Nada

VER: |[DBASE](#) |[DBASS](#) |DFICB |[NOME\\_DAT](#) |[PATH\\_PAN](#)

EJEMPLO:

|FICHEROS;

clientes#0;

ctcuen1 #1;

|FIN;

|[PROCESO](#) altacon;

|[DBASS](#) dirbase;

dirfic =dirbase+"conta4/fich/"+empresa+"/";

|[PATH\\_DAT](#) #1 dirfic;

|[ABRE](#) #0;

|[ABRE](#) #1;

|[LEE](#) 020#0=;

#1#0 = #0#0;

|[GRABA](#) 020#1n;

|[CIERRA](#) #0;

|[CIERRA](#) #1;

|FPR1C;

## **PATH\_PAN**

SINTAXIS: |PATH\_PAN #n variable alfanumérica;

DONDE: n es el número de fichero.

PROPÓSITO: Poner el contenido de la variable alfanumérica como directorio de todas las pantallas del fichero n. La variable no debe exceder en más de 42 caracteres y los directorios se deben separar con '/'. No se debe usar '\' o 'backslash' .

DEVUELVE: Nada

VER: [|DBASS](#) [|DBASE](#) [|DPAN](#) [|PATH\\_DAT](#)

EJEMPLO: Dado una aplicación /u/ds/gest/

[|PATH\\_PAN](#) directorio;

|| en directorio contendrá "/u/ds/gest/pan/".

## PAUSA

SINTAXIS: |PAUSA;

PROPÓSITO: Aparece el mensaje 'Pulse tecla' en la línea 24 y el programa se detiene en espera de lo propio.

DEVUELVE:

FSalida >= 0 ,devuelve un código de salida especial para las teclas de control.

0 = Return

1 = Escape

2 = Flecha Arriba

3 = Flecha Abajo

4 = Página Arriba

5 = Página Abajo

6 = Consulta

7 = Abortar (CTRL-C)

8 = Ayuda (HELP)

9 = Función 1

10= Función 2

11 a n = Función 3 a n (hasta 18 funciones)

VER: [|ENCAMPO](#) [|CONFI](#) [|MENSA](#)

EJEMPLO:

|[PROCESO](#) decide;|TIPO 0;

|[MENSA](#) "0000Continuar RETURN sino ESCAPE";

|[PAUSA](#);

|[SI](#) FSalida ! 1;|Y FSalida ! 7;

....

|[FINSI](#);

|FPRC;

## **PDEFECTO**

SINTAXIS: |PDEFECTO fichero,desdecampo,hastacampo;

PROPÓSITO: Pone en los campos comprendidos desde (inclusive) hasta (no inclusive) el valor por defecto indicado en el DEF. Si algún valor del desde o hasta no es válido hará un [DDEFECTO](#).

Si FSalida < 0 error.

EJEMPLO:

|[PROCESO](#) pdefecto;

|[ABRE](#) #1;

#1#0 = #0#0;

|[LEE](#) 040 #1=;

|[SI](#) FSalida=0;

|[PDEFECTO](#) #1,100,199;

|[GRABA](#) 040#1;

|[FINSI](#);

|[CIERRA](#) #1;

|FPRC

## **PIEINF**

SINTAXIS: |PIEINF;

PROPÓSITO: Imprime el pie del informe, quedando lista la impresión para el siguiente impreso.

DEVUELVE:

FSalida = 0 La impresión del pie se ha desarrollado satisfactoriamente.

FSalida = -1 Se ha producido algún error.

VER: [|PRINT](#) [|INFOR](#) [|FININF](#)

EJEMPLO:

[|PROCESO](#) imprimir;|TIPO 0;

[|IMPRE](#) 0;

[|SI](#) FSalida ] 0;

[|INFOR](#) informe;

[|SI](#) FSalida ] 0;

[|PRINT](#);

[|PIEINF](#);

[|FINSI](#);

[|FININF](#);

[|FINSI](#);

[|FINIMP](#);

|FPRC;

## **PINDA**

SINTAXIS: |PINDA #n#f;

DONDE:

n: Número de pantalla

f: Número de fichero

PROPÓSITO: Pinta los datos de la pantalla n del fichero f;

DEVUELVE: Nada.

VER: |[PINPA](#) |[DDEFECTO](#)

EJEMPLO:

|FICHEROS;

control#0;

|FIN;

|PROGRAMA;

|[CLS](#);

|[CABEZA](#) "FICHERO DE CONTROL";

|[PINPA](#) #0#0;

|[ABRE](#) #0;

|[LEE](#) 120#0p;

|[PINDA](#) #0#0; || Pinta los datos de la pantalla 0 del fichero

control.

|[ENDATOS](#) #1#0;

|[GRABA](#) 020#0a;

|[LIBERA](#) #0;

|[CIERRA](#) #0;

|FPRO;

## **PINPA**

SINTAXIS: |PINPA #n#f;

DONDE:

n: Número de pantalla

f: Número de fichero.



PROPÓSITO: Pinta la pantalla n del fichero f;

DEVUELVE: Nada.

VER: [|ENDATOS](#) [|PINDA](#) [|MANTE](#)

EJEMPLO:

```
|FICHEROS;  
    clientes#0;  
|FIN;  
|PROGRAMA;  
    |CLS;  
    |CABEZA "ESTE ES EL TITULO";  
    |PINPA #0#0; || Pinta la pantalla 0 del fichero de clientes.  
    |MANTE #P0;  
|FPRO;
```

## PINTA

Esta sentencia posee dos sintaxis que realizan dos acciones diferentes:

SINTAXIS 1: [|PINTA](#) [variable numérica,] campo o variable;

PROPÓSITO: Pinta un campo o variable en pantalla. Se puede indicar en qué posición se ha de pintar con una variable o constante numérica, esto es obligatorio en caso de ser una variable lo que se ha de pintar, en caso de un campo si no se especifica la posición se asume la del def.

DEVUELVE: Nada.

SINTAXIS 2: [|PINTA](#) variable numérica;

PROPÓSITO: Posiciona el cursor en la posición indicada por la variable o constante.

DEVUELVE: Nada.

VER: [|ATRI](#) [|GRAF](#)

EJEMPLO:

Para la sintaxis 1:

|[PINTA](#) 1020 , "Hola!";

|[PINTA](#) #0#1;

Para la sintaxis 2:

|[PINTA](#) 1020; || Cursor en 1020 para |[GRAF](#) u otra similar.

## PINVE

SINTAXIS: |PINVE p,q,nombre;

DONDE:

py q: Posiciones inicial y final de la pantalla calculadas a partir de la expresión: ***fila\*100+columna***

nombre: Nombre del fichero donde vamos a pintar la ventana.

PROPÓSITO: pintar la ventana, grabada en disco, "nombre" en la pantalla.

DEVUELVE:

FSalida = 0 Si la captura de la ventana del disco se ha realizado con éxito.

FSalida = -1 Si se ha producido algún error.

VER: |[GRABAVE](#) |[PUSHV](#) |[POPV](#)

EJEMPLO:

|[PINVE](#) 1010,2070,fichero;

|| Pintará la ventana "fichero" desde la posición (10,10) a la (20,70).

## PONREGN

SINTAXIS: |PONREGN #n, varocamnum;

DONDE:

n: Número de fichero a tratar.

Varocamnum: Variable o campo (con representación #X#Y).

PROPÓSITO: Asigna el contenido de varocamnum al registro interno actual. Esta sentencia sólo se puede ejecutar tras activar la clave de búsqueda con la sentencia |SELECT #0#n; de lo contrario su efecto se ignora o puede provocar una pérdida de los resultados de una lectura por clave.

DEVUELVE: Nada.

VER: [|DAREGN](#)

## POPV

SINTAXIS: |POPV;

PROPÓSITO: Repone la zona de la pantalla guardada con el último [|PUSHV](#) empleado.

DEVUELVE: Nada

VER: [|PUSHV](#) [|PINPA](#)

EJEMPLO:

[|PROCESO](#) ventana;|TIPO 0;

[|PUSHV](#) 1010 2070; || coincide con el tamaño de la pantalla a pintar

[|PINPA](#) #0#1;

[|ENDATOS](#) #1#1;

[|POPV](#);

|| Tras provocar el alta del fichero 1 reponemos la pantalla que habíamos salvado

|| con la sentencia [|PUSHV](#) desde la posición 10,10 a 20,70.

|FPRC;

## PRINT

SINTAXIS: |PRINT;

PROPÓSITO: Sentencia la impresión de una línea de informe, si es la primera de la hoja se imprime previamente la cabecera. Si antes de ejecutar el |PRINT, la variable FSalida contiene la palabra "PÁGINA", se fuerza un salto de página.

DEVUELVE:

FSalida = 0 Si la impresión de la línea ha sido correcta.

FSalida = -1 Caso de producirse algún error de impresión.

VER: [|PIEINF](#) [|INFOR](#) [|FININF](#)

EJEMPLO:

```
|PROCESO imprimir;|TIPO 0;
|IMPRE 0;
|SI FSalida ] 0;
|INFOR informe;
|SI FSalida ] 0;
|PRINT; || Imprime una línea de INFORME.
|PIEINF;
|FINSI;
|FININF;
|FINSI;
|FINIMP;
|FPRC;
```

## PRINTA\_PAN

Esta instrucción funciona a partir del básico 9.9 T-7.

SINTAXIS: [|PRINTA\\_PAN](#) #numpan#fich,modo;

DONDE:

modo = 0 devuelve las dimensiones de la pantalla en formato alto\*100+ancho

modo < 0 salta -modo líneas (imprime todas)

modo > 0 imprime hasta como máximo modo (poner 9999 por ejemplo si no se quiere límite).

## PROCESO

EJEMPLO:

|PROCESO cualquiera;

nDeci\_Imp = 2;

#mificha#importe = a / b; || redondeado a 2 decimales

|[PINTA](#) #mificha#importe; || sale con dos decimales

nDeci\_Imp = 1;

#mificha#importe = a / b; || redondeado a 1 decimal

|[PINTA](#) #mificha#importe; || sale con un decimal

|FPRC;

## PTEC

SINTAXIS: |PTEC <variable numérica>;

PROPÓSITO: Pone en el buffer de teclado la tecla cuyo número ASCII o número de control contiene la variable numérica. El número de control es 800 más el número ordinal de tecla que le corresponde según la entrada de teclas en el programa de configuración de terminales. Posee un buffer interno de 200 pulsaciones.

DEVUELVE: Nada.

VER: |[ENCAMPO](#) |[ENDATOS](#) Operación ?

EJEMPLO:

|[PROCESO](#) entra;|TIPO 0;

```
|SI saltar = 1;  
    |PTEC 801; || Return  
|FINI;  
|ENCAMPO #5#0;  
|FPRC;
```

## PULSATECLA

SINTAXIS: |PULSATECLA;

PROPÓSITO: Detecta qué tecla ha sido pulsada. No es afectada por |PTEC.

DEVUELVE:

FSalida <> -1 Recoge la tecla pulsada .

FSalida = -1 En caso de no haberse pulsado ninguna tecla.

EJEMPLO:

```
|PROCESO atención;|TIPO 0;  
    |PULSATECLA;  
    |SI FSalida = " "; || La tecla pulsada es el espacio en  
    blanco.  
        |MENAV "0000Proceso detenido!";  
    |FINI;  
|FPRC;
```

## PUSHV

SINTAXIS: |PUSHV <variable numérica>;

DONDE:

Variable numérica tiene el formato *ficiffcl*, donde *fi* es fila inicial, *ci* columna inicial, *ff* fila final y *cf* columna final.

PROPÓSITO: Guarda en memoria una zona de la pantalla comprendida entre la posición inicial y final contenidas en la variable numérica.

PUSHV funciona como una pila, esto es, la última pantalla que guardamos (hasta 7 como máximo) es la primera que recuperamos siendo la primera que guardamos la última que recuperamos. Se les suele denominar elementos de almacenamiento “ LIFO “.

DEVUELVE: Nada.

VER: |[POPV](#) |[PINPA](#)

EJEMPLO:

|[PROCESO](#) ventana;|TIPO 0;

|[PUSHV](#) 1010 2070; || coincide con el tamaño de la pantalla a pintar.

|[PINPA](#) #0#1;

|[ENDATOS](#) #1#1;

|[POPV](#);

|FPRC;

## QBF

SINTAXIS: |[QBF](#) <variable alfanumérica>;

PROPÓSITO: Quita los espacios en blanco que hay después del ultimo carácter no espacio contenido en la variable alfanumérica.

DEVUELVE: Nada.

VER: |QBT

EJEMPLO:

alfa = "Hola ";

|[QBF](#) alfa;

|| alfa queda = "Hola"

## QBT

SINTAXIS: |QBT <variable alfanumérica>;

PROPÓSITO: Quita todos los espacios en blanco contenidos de la variable alfanumérica.

DEVUELVE: Nada.

VER: |QBF

EJEMPLO:

```
alfa = "Hola que tal ";
|QBT alfa;
|| alfa queda = "Holaquetal";
```

## QUE\_SISTEMA

SINTAXIS: |QUE\_SISTEMA varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Permite conocer el sistema operativo con que ha sido creado el AGIRUN y, en la mayor parte de los casos, coincidirá con el sistema operativo en uso. El tipo lo recogeremos en varalfa.

DEVUELVE: La variable varalfa podrá tomar los siguientes valores según sea el sistema operativo utilizado:

ESDOS	para D.O.S.
ESXENIX	para XENIX.
ESAIX	para AIX.
ESUNIX	para UNIX (genérico)

EJEMPLO:

```
|FICHEROS;
terminal #0;
```



```
|FIN;

|VARIABLES;

    sistema = "";

|FIN;

|PROGRAMA;

    |QUE_SISTEMA sistema;

    |SI sistema ! "ESDOS";

        || instrucciones ...

    |SINO;

        || instrucciones ...

    |FINSI;

|FPRO;
```

## **RATAR**

Como [|ATAR](#) pero en local.

## **RBASS**

Saca el directorio en la maquina local de instalación.

## **RCOMPARE**

Como [|COMPARE](#) pero en local.

## **RCOPIA\_FICHERO**

SINTAXIS:

```
|RCOPIA_FICHERO aAlfa1, aAlfa2;
```

Copiará el fichero aAlfa1 situado en la máquina cliente en aAlfa2 situándose en el cliente.

## **RDEFECTO**

SINTAXIS: |RDEFECTO #n;

DONDE: n un número de fichero.

PROPÓSITO: Pone los valores de los defecto (D) a otros ficheros indicados en el def del fichero n y que no corresponden (los otros ficheros) a ninguna relación del fichero n.

DEVUELVE: Nada.

VER: |DEFECTO

EJEMPLO:

|RDEFECTO #0;

## **RDELTREE**

Como |DELTREE pero en local.

## **RDESCOMPRIME**

Como |[DESCOMPRIME](#) pero en local.

## **RDETAR**

Como |DETAR pero en local.

## **REFRESCACONTROL**

SINTAXIS: |REFRESCACONTROL handle;

PROPÓSITO: Se le pasa el handle del control a refrescar (numérico). Actúa según la naturaleza del control, en el caso del grid lo reinicializa (se vuelven a cargar los límites y se repintan).

## RENOMBRA\_FICHERO

SINTAXIS: RENOMBRA\_FICHERO alfa1,alfa2;

PROPÓSITO: Renombra un fichero.

Alfa1 = viejo nombre, alfa2 = nuevo nombre.

Si FSalida < 0 no se pudo renombrar.

EJEMPLO:

## REPINTA\_TEXTO

SINTAXIS: |REPINTA\_TEXTO línea, columna;

DONDE:

línea es una variable numérica cuyo valor expresa el número de línea donde se va a repintar el texto.

columna es una variable numérica que indicará el número de columna donde se repintará el texto.

PROPÓSITO: La sentencia |REPINTA\_TEXTO permite repintar texto, en operaciones de edición del fichero de textos, en la posición dada por línea y columna. Si línea < 0 partirá de la posición línea = 0 y en caso de asignarle un valor superior al máximo, línea tomará dicho valor máximo. Análogamente para el caso de columna.

DEVUELVE: Nada.

VER: [|EDITA](#) [|F\\_TEXTO](#)

## RFABRE

Como [|FABRE](#) pero en local.

## **RFCIERRA**

Como |[FCIERRA](#) pero en local.

## **RFGRABA**

Como |[FGRABA](#) pero en local.

## **RFLEE**

Como |[FLEE](#) pero en local.

## **RGRABA\_NUMERO**

Como |[GRABA\\_NUMERO](#) pero en local.

## **RLEE\_NUMERO**

Como |[LEE\\_NUMERO](#) pero en local.

## **RMDIR**

SINTAXIS: RMDIR alfa1;

PROPÓSITO: En alfa1 el directorio a borrar, de momento solo borra directorios vacíos (rd o rmdir standar) pero en cualquier momento tendrá la eficacia “borradora” del rm -r.

Si FSalida < 0 error.

EJEMPLO:

## **RMKDIR**

Como |[MKDIR](#) pero en local.

## **R\_PDIR**

Como |[\\_PDIR](#) pero en local.

## **R\_SDIR**

Como |[\\_SDIR](#) pero en local.

## **RSYSTEM**

SINTAXIS: |RSYSTEM "mandato";

Igual que el |SYSTEM pero se ejecuta 'preferentemente' en el cliente.  
En caso de ser la conexión cliente servidor el system se ejecuta en el cliente y se devuelve FSalida con la ip del cliente. En caso de ejecutarse en el servidor (local) se devuelve FSalida vacío.  
(Ojo el |SYSTEM no toca el FSalida para nada).

## **RTEMPO**

SINTAXIS: |RTEMPO alfa;

Devuelve un nombre de fichero temporal en el cliente si C/S.

## **SAL**

SINTAXIS: |SAL;

PROPÓSITO: Permite abandonar un bucle de instrucciones siendo la próxima instrucción a ejecutar, la siguiente a la sentencia |SIGUE.

DEVUELVE: Nada

EJEMPLO:

```
|FICHEROS;  
    terminal #0;  
|FIN;
```

```

|VARIABLES;

    arrai = 0;

    ndat = 0;

|FIN;

|PROGRAMA;

    |PARA;|SI;|HACIENDO;

        || instrucciones ...

        |BMENUG arrai,0,2,ndat,0,0;

        |SI FSalida < 1;

            |SAL;

            |FINSI;

            || instrucciones ...

        |SIGUE;

        || instrucciones ...

|FPRO;

```

## SALTO\_DE\_LINEAS

SINTAXIS: |SALTO\_DE\_LINEAS num;

A ejecutar en un TIPO 5, indica los saltos entre líneas (normal 1)

## SCROLL

SINTAXIS: |SCROLL varnum1,varnum2,varnum3;

DONDE: varnum1 , varnum2 , varnum3 campos numéricos.

PROPÓSITO: 'Mueve ' la pantalla comprendida entre los límites de varnum1 y varnum2 (en formato LLCC) tantas líneas según varnum3 (en positivo hacia abajo y en negativo hacia arriba)

**NOTA:** esta función va bien pero es la primera cabeza de puente para la implementación de los scroll en todo el entorno así que es de utilidad limitada de momento.

EJEMPLO:

## SELECT

SINTAXIS: |SELECT #c#f;

DONDE: c el número de clave a seleccionar y f el número de fichero.

PROPÓSITO: Selecciona la clave c del fichero f que se usará en todas las siguientes lecturas del mismo. En caso de c = 0, el acceso al fichero será directo por número de registro utilizando las sentencias [|PONREGN](#) y [|DAREGN](#).

DEVUELVE:

FSalida = 0 La selección ha sido correcta.

FSalida > 0 Se ha producido algún error de indexado o del sistema

VER: [|LEE](#) [|ABRE](#) [|CIERRA](#) [|PONREGN](#) [|DAREGN](#)

EJEMPLO:

|SELECT #2#0;

[|LEE](#) 020#0p; || usa clave 2 para la lectura.

|SELECT #3#0;

[|LEE](#) 020#0p; || usa clave 3 para la lectura.

## SI

SINTAXIS: |SI condición;

DONDE: condición es la restricción impuesta.

VER: MANUAL DE PROCESOS: BIFURCACIONES CONDICIONALES |SINO [|FINSI](#)

EJEMPLO:

```

|PROCESO imprimir;|TIPO 0;

|IMPRE 0;

|SI FSalida ] 0;

|INFOR informe;

|SI FSalida ] 0;

|BUCLE nombucle;

|FINSI;

|FININF;

|FINSI;

|FINIMP;

|FPRC;

```

## SIALARMA

SINTAXIS: |SIALARMA;

PROPÓSITO: Activa la posibilidad de interrupción de un proceso por la activación de la alarma, después de haber interrumpido esta posibilidad mediante [NOALARMA](#).

DEVUELVE: Nada.

VER: |[ALARMA](#) |SIALARMA

EJEMPLO:

```

|FICHEROS;

    datos #0;

|FIN;

|PROGRAMA;

|NOALARMA;

|| mientras se este ejecutando este programa no se activara

```

la alarma



|| instrucciones ...

|SIALARMA;

|FPRO;

## SIGUE

SINTAXIS: |SIGUE;

VER: MANUAL DE PROCESOS: BUCLE DE INSTRUCCIONES [|PARA](#) [|SI](#) [|HACIENDO](#).

EJEMPLO:

[|PROCESO](#) raya;|TIPO 0;

[|GRAF](#) -1;

[|PARA](#) i =1170;|[|SI](#) i < 2070;|[|HACIENDO](#) i=i+100;

[|PINTA](#) i;

[|GRAF](#) 0;

[|SIGUE](#);

[|GRAF](#) -2;

|FPRC;

## SINO

SINTAXIS: |SINO;

VER: MANUAL DE PROCESOS: BIFURCACIONES CONDICIONALES [|SI](#) [|FINSI](#)

EJEMPLO:

[|PROCESO](#) imprimir;|TIPO 0;

[|IMPRE](#) 0;

[|SI](#) FSalida ] 0;

[|INFOR](#) informe;

```
|SI FSalida ] 0;  
    |BUCLE nombucle;  
    |FINSI;  
    |FININF;  
|SINO;  
    |ERROR;  
    |ACABA;  
    |FINSI;  
    |FINIMP;  
|FPRC;
```

## SIPARA

SINTAXIS: |SIPARA;

PROPÓSITO: Permite que todos los bucles declarados (DEFBUCLE), ejecutados a partir de esta sentencia, puedan ser detenidos y abortados desde teclado.

DEVUELVE: Nada

VER: |BUCLE

EJEMPLO:

```
|PROCESO imprimir;|TIPO 0;  
    |IMPRE 0;  
    |SI FSalida ] 0;  
        |INFOR informe;  
    |SI FSalida ] 0;
```

|SIPARA; || Podremos detener la impresión de la ordenación del fichero por su bucle declarado.

```
|BUCLE nombucle;
```

[|FINSI;](#)

[|FININF;](#)

[|FINSI;](#)

[|FINIMP;](#)

|FPRC;

## SLEEP

SINTAXIS: |SLEEP <variable numérica>;

PROPÓSITO: Hace una pausa de varios segundos, tantos como se indica en variable numérica.

DEVUELVE: Nada.

EJEMPLO:

|SLEEP 5; || Espera 5 segundos

## SQL

SINTAXIS : |SQL varalfa

DONDE: varalfa es un string que contiene una sentencia SQL.

varalfa puede contener una sentencia con una “@” en cuyo caso cogerá el contenido del fichero que le sigue a la arroba;

P.ej.

Es decir, si voy a usar un fichero ,éste contendrá lo siguiente :

```
/u/tmp/query.txt ---> agifa071.1,agifa071.2,agifa010.3 FROM  
agifa071,agifa010 INTO pepe.def WHERE agifa010.10 == 0
```

entonces puedo hacer:

```
aAlfa = "SELECT @/u/tmp/query.txt"  
|SQL aAlfa;
```

PROPOSITO : Obtiene un conjunto de registros de la fuente solicitada en el fichero destino indicado en la sentencia SQL que cumplan la(s) condición(es) .

DEVUELVE :

En FSalida se devuelven dos resultados en un string:

En las primeras posiciones se devuelve el resultado de la operación

= 0 si todo ha ido bien

< 0 si ha habido algún problema

Si este primer resultado es = 0, el string va seguido de un literal 'Seleccionados : XXX' donde XXX es el numero de registros que

se han obtenido de la consulta

Ejemplo :

De momento la sentencia tiene las siguientes normas a seguir para su buen funcionamiento:

- \* La primera parte de la cadena declara los campos que se obtendrán en el nuevo def de las siguientes formas :

"SELECT agifa024.2,agifa024.3 FROM" de forma que se detallan el nº de campos. OJO , no dejar espacios entre las comas.

"SELECT \* FROM" de forma que el nuevo def hereda todos los campos del/de los fichero/s fuente

- \* La fuente de los datos serán uno o dos ficheros y deben estar declarados en el apartado '|FICHEROS'.

Si son dos ficheros deberán ser del tipo cabecera/líneas o deben tener clave similar. Irán separados por una coma y el fichero

'principal' será el que está en primer lugar y el def resultante hereda la clave primaria del fichero 'principal'. No se necesita

declarar en la cláusula 'SELECT' los campos que conforman dicha clave, ya que la sentencia los incluye automáticamente.

- \* Se pueden usar dos nomenclaturas para indicar el fichero destino que contendrá los datos.

- "INTO fichero.def" con lo que el runtime generará un def en tiempo real conteniendo los datos del resultado de la consulta (el cual

deberá ser cargado con un CARGA\_DEF).

- "INTO fichero" con lo que el runtime sólo generará un fichero .dat con el resultado (el cual habrá que apuntarlo con un NOME\_DAT)

\* La cláusula 'WHERE' se usa indicando el nombre del fichero.nº de campo + operador + operando (WHERE agifa024.2 == 125) o también cuando la fuente de datos del JSQL es un solo fichero, nº de campo + operador + operando (WHERE 2 == 125) y usa sólo cuatro operadores :

- LIKE, se cumple cuando el campo contiene el substring indicado después.

P.ej.

JSQL "SELECT \* FROM agifa024 INTO tempo.def WHERE 1  
LIKE 'Perez'"

Cliente 0 Nombre Pepe Pérez -----> Verdadero

Cliente 1 Nombre Manuel López -----> Falso

Cliente 2 Nombre Roberto Perezito -----> Verdadero

- '=', se cumple cuando el campo comienza por el substring indicado después.

P.ej.

JSQL "SELECT agifa071.1,agifa080.4,agifa080.5 FROM  
agifa071,agifa080 INTO tempo.def WHERE agifa071.1 = 'Pepe'"

Cliente 0 Nombre Pepe Pérez -----> Verdadero

Cliente 1 Nombre Manuel López -----> Falso

Cliente 2 Nombre Roberto Perezito -----> Falso

- '==', se cumple cuando el campo es igual al substring indicado después.

P.ej.

|SQL "SELECT \* FROM agifa024 INTO tempo.def WHERE 1 == 'Manuel Lopez'"

Cliente 0 Nombre Pepe Pérez -----> Falso

Cliente 1 Nombre Manuel López -----> Verdadero

Cliente 2 Nombre Roberto Perezito -----> Falso

- 'BETWEEN', se cumple cuando el campo se encuentra en el intervalo indicado después.

P.ej.

|SQL "SELECT \* FROM agifa071,agifa080 INTO tempo.def WHERE agifa080.0 BETWEEN 0,1"

Cliente 0 Nombre Pepe Pérez -----> Verdadero

Cliente 1 Nombre Manuel López -----> Verdadero

Cliente 2 Nombre Roberto Perezito -----> Falso

\* Se pueden encadenar cláusulas WHERE, pero solo mediante los operadores lógicos 'AND' u 'OR'

P.ej.

|SQL "SELECT \* FROM agifa024 INTO tempo.def WHERE 0 BETWEEN 0,1 AND 1 = 'Pepe'"

Cliente 0 Nombre Pepe Pérez -----> Verdadero

Cliente 1 Nombre Manuel López -----> Falso

Cliente 2 Nombre Roberto Perezito -----> Falso

Cliente 3 Nombre Pepe Martínez -----> Falso

\* No existe (de momento) el ORDER BY

**NOTA IMPORTANTE**

Cuando se hace un SQL de con un solo fichero en la cláusula 'FROM', el nuevo def tendrá la estructura del def declarado, es decir, se copian las relaciones y las claves ya que lo que hace es una copia de la estructura del fichero original. Sin embargo si se declaran más de un fichero en 'FROM', el nuevo def tendrá como estructura los campos de la clave primaria del fichero principal (el primero declarado en 'FROM'), añadiendo los campos que se hayan declarado en la cláusula 'SELECT'. Pero ojo porque las relaciones que tuviesen esos campos con otros ficheros **SE PIERDEN**.

## SUB\_EJECUTA

SINTAXIS: |SUB\_EJECUTA varalfa;

DONDE: varalfa es una variable alfanumérica.

PROPÓSITO: Ejecuta el fichero ejecutable Agi especificado en varalfa en MODO SUBORDINADO. Durante este tipo de ejecución se carga el ejecutable Agi junto al que está actualmente procesándose, dejando en 'stand-by' al fichero Agi principal que, tras finalizar la ejecución en modo subordinado, proseguirá normalmente con la aplicación.

DEVUELVE: En FSalida el factor de control de subordinado.

EJEMPLO:

```
|FICHEROS;

      clientes#0;

|FIN1;

|VARIABLES;

      prg1="cliente1.Agi";

      prg3="cliente2.Agi";

      {-1} menú = "";

      menu1 = "2111";

      menu2 = "1";

      menu3 = " Elija Opción --> ";
```

```

menu4 = "VC";

menu5 = " Ventas ";

menu6 = " Compras ";

|FIN;

|PROGRAMA;

|PARA;|SI FSalida > 0;|HACIENDO;

|CLS;

|CABEZA "Elección de Programa";

|ATRI I;

|MENU menú;

|ATRI 0;

|SI FSalida = 1; || primera opción de menú

|CLS;

|SUB_EJECUTA prg1;

FSalida = 1;

|FINSI;

|SI FSalida = 2; || segunda opción de menú

|CLS;

|SUB_EJECUTA prg2;

FSalida = 1;

|FINSI;

|SIGUE;

|FPRO;

```

## SUB\_EJECUTA\_NP

SINTAXIS: |SUB\_EJECUTA\_NP programa;



programa -> alfa

PROPÓSITO: Es igual que el SUB\_EJECUTA pero no crea la nueva pestaña.

## **SUB\_PINPA**

SINTAXIS: |SUB\_PINPA #numpan#fichero, idpantalla;

PROPÓSITO: Igual que PINPA pero en idpantalla (numérica) va el id de la pantalla de la cual la que creamos (o activamos) es 'hija'.

DEVUELVE: Tanto los PINPA como SUB\_PINPA devuelven en FSalida el id de la pantalla.

## **SYSTEM**

SINTAXIS: |SYSTEM <variable alfanumérica>.

PROPÓSITO: Ejecuta el comando del S.O. contenido en la variable alfanumérica. Con el Run-Time o básico AGI vienen una serie de interfaces para copiar fichero, crear directorios, etc..., que permiten usar funciones comunes de los sistemas operativos con una sintaxis común independiente del S.O.

DEVUELVE: Nada

EJEMPLO:

```
|PROCESO hazdir;|TIPO 0;  
    alfa = "/Agi/bin/Agimkdir " + directorio;  
    |SYSTEM alfa;  
|FPRC;
```

## **TECLA\_FUNCION**

SINTAXIS: |TECLA\_FUNCION varnum1,varnum2,varalfa;

PROPÓSITO: Programa (varnum2 = 0) o Recoge la programación (varnum2 = 1) de la tecla de función varnum1 (puede ser de 0 a 19) de varalfa.

DEVUELVE:

Convenio de programación de las teclas de función:

Si la tecla de función contiene string vacío devolverá su código interno de tecla.

Si contiene un string:

Si el primer caracter es '@' el resto del string se pone en el buffer de teclado y las siguientes lecturas del teclado iran devolviendo ese string hasta que se acabe.

Si no es '@' se llama a una función de interrupción si esta definida (si no se devuelve el código interno) pasando el string que debe tener un convenio segun la función de interrupción, caso mdrun (el normal y único posible desde un mdr):

Primer caracter '|': ejecuta el mdr en modo subordinado especificado detrás del '|'.

Primer caracter '&': ejecuta un PROCESO declarado externo del mdr actual especificado detrás del '&'. Se ejecuta si el nombre indicado se encuentra dentro de la tabla de procesos externos (declarados con un '&') sino pita.

Otro Primer caracter : no tiene efecto ... pita.

EJEMPLO:

## TEMPO

SINTAXIS: |TEMPO <variable alfanumérica>;

PROPÓSITO: Devuelve en variable alfanumérica el nombre de un fichero temporal quedando reservado hasta que no se borre con [FBORRA](#).

DEVUELVE:

FSalida = 0 Si la operación de generar un fichero temporal ha sido correcta.

FSalida = -1 En caso de presentarse algún error durante dicha acción.

VER: [|GRABAVE](#) [|FABRE](#) [|FBORRA](#)

EJEMPLO:

```
||PROCESO grabave;|TIPO 0;  
    || Graba una pantalla sin peligro de  
    || que otro usuario la 'machaque'  
    |TEMPO fichero;  
    ||GRABAVE 1010,2070,fichero;  
    ...  
    ||FBORRA fichero;  
|FPRC;
```

## VENTANA

SINTAXIS: |VENTANA pi, pf, -1, -1, “”;

PROPÓSITO: Permite crear una ventana sin que destruya la pantalla principal pi y pf (numéricas) marco aplicable a la ventana.

Los otros parámetros están para un futuro.

DEVUELVE: FSalida devuelve el ID de la ventana; nVentana = FSalida.

## VERSIONRTME

SINTAXIS: VERSIONRTME alfa1;

PROPÓSITO: En alfa1 devuelve la versión del ds-system como sale en pantalla: Ejm “5.01a”

EJEMPLO:

## VETE

SINTAXIS: |VETE nometiq;

DONDE: nometiq es una variable alfanumérica con que etiquetaremos la operación a considerar.

PROPÓSITO: Bifurca incondicionalmente al nombre de etiqueta definido en [|ET](#).

DEVUELVE: Nada.

VER: MANUAL DE PROCESOS: BIFURCACIONES INCONDICIONALES [|ET](#)

EJEMPLO:

```
|PROCESO salto;|TIPO 0;
```

```
...
```

```
|ET punto1;| A este punto volveremos siempre que la  
variable FSalida sea 7.
```

```
...
```

```
|SI FSalida = 7;
```

```
|VETE punto1; || Salta a donde se encuentre la  
etiqueta punto1.
```

```
|FINSI;
```

```
|PARA i = 1;|SI i < 7;|HACIENDO i = i + 1;
```

```
...
```

```
|SI FSalida = 7;
```

```
|VETE punto1;
```

```
|FINSI;
```

```
...
```

```
|SIGUE;
```

```
|FPRC;
```

## WORD\_ABRE

SINTAXIS: |WORD\_ABRE nWord, aAbre;

DONDE: nWord es una variable numérica con la que identificaremos al fichero de Word, aAbre es una variable alfanumérica que guarda el path del fichero.

PROPÓSITO: Abre un documento de Word.

DEVUELVE: Nada.

VER: [|WORD ASIGNA CAMPO](#) [|WORD ASIGNA TEXTO](#)  
[|WORD CARGA TEXTO](#) [|WORD IMPRIME](#) [|WORD PROPIEDADES](#)  
[|WORD SALVA](#)

EJEMPLO:

```
aAlfa = "C:/MODELOS";          |RMKDIR aAlfa;

aAlfa = "C:/MODELOS/DOCUMENTOS";    |RMKDIR aAlfa;

nRegistro = 0;

aOrigen    = eaRutaDocumento + "/" + #111#4 + "." + eaExtension;
|QBT aOrigen;

aDestino    = "C:/MODELOS/DOCUMENTOS/" + #111#4 + "." +
eaExtension; |QBT aDestino;

|RCOPIA\_FICHERO aOrigen, aDestino;

|SI FSalida ! 0;

    |MENAV "    No se ha podido copiar el Documento al Directorio
de Trabajo";

    |ACABA;

|FINSI;

nWord = 1;

aAbre = aDestino;

|MENSA "    Abriendo el Documento, espere un Momento.";
|WORD_ABRE nWord, aAbre;
```

## WORD\_ASIGNA\_CAMPO

SINTAXIS: |WORD\_ASIGNA\_CAMPO nWord, aCampo, aAlfa;

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto, aCampo es una variable alfanumérica con la que guardaremos el campo y aAlfa es una variable alfanumérica que guarda el texto que asignaremos al documento.

PROPÓSITO: Igual que |WORD\_ASIGNA\_TEXTO pero pasándole el campo.

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_TEXTO](#) [|WORD\\_CARGA\\_TEXTO](#)  
[|WORD\\_IMPRIME](#) [|WORD\\_PROPIEDADES](#) [|WORD\\_SALVA](#)

EJEMPLO:

|WORD\_ASIGNA\_CAMPO nWord, "#18", aAlfa;

## WORD\_ASIGNA\_TEXTO

SINTAXIS: |WORD\_ASIGNA\_TEXTO nWord, aAsigna, aAlfa;

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto, aAsigna es una variable alfanumérica con la que identificaremos al documento de Word y aAlfa es una variable alfanumérica que guarda el texto que asignaremos al documento.

PROPÓSITO: Asigna un texto a un documento de Word.

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_CAMPO](#) [|WORD\\_CARGA\\_TEXTO](#)  
[|WORD\\_IMPRIME](#) [|WORD\\_PROPIEDADES](#) [|WORD\\_SALVA](#)

EJEMPLO:

FSalida = 0;

aAlfa = #999#2;

[|PARA](#); [|SI](#) FSalida = 0; [|HACIENDO](#);

aAsigna = "!" + #999#1;

|WORD\_ASIGNA\_TEXTO nWord, aAsigna, aAlfa;

[SIGUE;](#)

## **WORD\_CARGA\_TEXTO**

SINTAXIS: |WORD\_ASIGNA\_TEXTO nWord, aAsigna, aDestino;

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto, aAsigna es una variable alfanumérica con la que identificaremos al documento de Word y aDestino es el fichero que tiene lo que queremos incrustar en el fichero de Word.

PROPÓSITO: Inserta el contenido de un fichero en un documento de Word.

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_CAMPO](#) [|WORD\\_ASIGNA\\_TEXTO](#)  
[|WORD\\_IMPRIME](#) [|WORD\\_PROPIEDADES](#) [|WORD\\_SALVA](#)

EJEMPLO:

aDestine = "/ds/impresio/" + "fichero.txt";

aAsigna = "!" + "&EXTR&"; ||Extracto.....

|WORD\_CARGA\_TEXTO nWord, aAsigna, aDestine;

## **WORD\_IMPRIME**

SINTAXIS: |WORD\_IMPRIME nWord, aAlfa;

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto, aAlfa es una variable alfanumérica con la que le diremos si sacamos el cuadro de diálogo o no al imprimir.

PROPÓSITO: Imprime un documento de Word mostrando o no el cuadro de dialogo.

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_CAMPO](#) [|WORD\\_ASIGNA\\_TEXTO](#)  
[|WORD\\_CARGA\\_TEXTO](#) [|WORD\\_PROPIEDADES](#) [|WORD\\_SALVA](#)

EJEMPLO:

|WORD\_IMPRIME nWord, "Dialogo"; ||imprime el fichero mostrando el cuadro de diálogo.

|WORD\_IMPRIME nWord, ""; ||imprime el fichero sin mostrar el cuadro de diálogo.

## WORD\_PROPIEDADES

SINTAXIS: |WORD\_PROPIEDADES nWord, aPropiedad, aModo;

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto, aPropiedad es una variable alfanumérica con la que le indicaremos la propiedad a modificar y aModo es una variable alfanumérica donde le diremos el valor de la propiedad.

PROPÓSITO: Modifica una propiedad de un documento Word dependiendo del contenido de aPropiedad y aModo. Estas variables pueden contener:

<u>aPropiedad</u>	<u>aModo</u>
Visible	SI / NO -> Pone visible o no
limpiafinal	"" (a Modo siempre es vacío) -> ajusta las páginas para eliminar las páginas en blanco del final)
Paginacion	aPagina -> Contiene el número de página a partir de la cual va a paginar.
Quit	"" (a Modo siempre es vacío) -> para salirse).

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_CAMPO](#) [|WORD\\_ASIGNA\\_TEXTO](#)  
[|WORD\\_CARGA\\_TEXTO](#) [|WORD\\_IMPRIME](#) [|WORD\\_SALVA](#)

EJEMPLO:



```
|WORD_PROPIEDADES nWord,"Visible", "NO";  
|WORD_PROPIEDADES nWord,"limpiafinal", "";  
|WORD_PROPIEDADES nWord,"Paginacion", aPagina;  
|WORD_PROPIEDADES nWord,"Quit", "";
```

## WORD\_SALVA

SINTAXIS: |WORD\_SALVA nWord, "";

DONDE: nWord es una variable numérica con la que identificaremos al Word abierto.

PROPÓSITO: Guarda un documento Word.

DEVUELVE: Nada.

VER: [|WORD\\_ABRE](#) [|WORD\\_ASIGNA\\_CAMPO](#) [|WORD\\_ASIGNA\\_TEXTO](#)  
[|WORD\\_CARGA\\_TEXTO](#) [|WORD\\_IMPRIME](#) [|WORD\\_PROPIEDADES](#)

EJEMPLO:

```
|WORD_SALVA nWord, "";
```

## XABRE

SINTAXIS: XABRE modo\_alfa,f ichero\_alfa, canal\_num;

DONDE:

modo\_alfa:

C = intentar abrir en el cliente si C/S

E = solo chequear existencia (FSalida = 0 no FSalida = 1 si).

B = modo binario

A = Añadir al final, si el fichero no existe lo crea.

W = grabar (machacar) si existe el fichero se lo carga sino lo crea

U = actualizar, el fichero debe existir

fichero\_alfa = path (y nombre) fichero.

canal\_num = variable donde viene el handle del fichero (excepto modo 'E')

DEVUELVE: FSalida < 0 no se abre sino el handle.

## **XCIERRA**

SINTAXIS: |XCIERRA canal\_num;

DONDE: canal\_num = handle del fichero a cerrar;

DEVUELVE: FSalida < 0 error

## **XLEE**

SINTAXIS: |XLEE canal\_num,max\_num,datos\_alfa;

DONDE:

canal\_num = handle del fichero;

max\_num = máximo caracteres a recibir;

datos\_alfa = alfanumérica que recibe los datos

DEVUELVE:

FSalida < 0 error

sino numero de caracteres leídos.

## **XGRABA**

SINTAXIS: |XGRABA canal\_num,datos\_alfa;

DONDE:

canal\_num = handle del fichero;

datos\_alfa = datos a grabar

DEVUELVE:

FSalida < 0 error

sino numero de caracteres grabados.

### **XLEE\_NUMERO**

SINTAXIS: |XLEE\_NUMERO canal\_num, tipo\_num, num;

DEVUELVE:

canal\_num = handle del fichero;

tipo\_num = 0 = Uchar , 1 Ushort , 2 Ulong , 3 float , 4 double;

num = el número a leer;

### **XGRABA\_NUMERO**

SINTAXIS: |XGRABA\_NUMERO canal\_num, tipo\_num, num;

DONDE:

canal\_num = handle del fichero;

tipo\_num = 0 = Uchar , 1 Ushort , 2 Ulong , 3 float , 4 double;

num = el número a grabar;

### **XLEEDEF**

SINTAXIS: |XLEEDEF canal\_num, #fichero;

DONDE:

canal\_num = handle del fichero;

fichero = def(sus datos) a cargar del fichero;

### **XGRABADEF**

SINTAXIS: |XGRABADEF canal\_num, #fichero;

DONDE:

canal\_num = handle del fichero;

fichero = def(sus datos) a grabar al fichero;

## **XPOSICION**

SINTAXIS: |XPOSICION canal\_num,posición\_num,modo\_num;

DONDE:

canal\_num = handle del fichero;

posición\_num = posición a poner

modo\_num = modo de la posición a poner:

0 relativo al origen del fichero

1 relativo a la posición actual

2 relativo al final del fichero

DEVUELVE:

En FSalida y posición\_num la posición relativa al origen del fichero.

Para saber la posición actual: posición\_num = 0 modo\_num = 1

## **\_PDIR**

SINTAXIS: \_PDIR alfa1;

PROPÓSITO: En alfa1 poner el directorio + comodines a leer ( de momento solo usar los asteriscos).

Si FSalida < 0 no hay entradas.

EJEMPLO:

## **\_SDIR**

SINTAXIS: \_SDIR alfa1;

PROPÓSITO: Mantener la misma alfa1 d \_PDIR o el \_SDIR ( con el resultado anterior)

Si FSalida < 0 no hay entradas.

**NOTA:** Siempre hay que leer hasta que de error.

## RECOGIDA DE DATOS DE UN DEF.

Para recoger datos de un DEF , se le pide al propio DEF pasándole en vez del número de campo, una variable alfanumérica como se especifica en los siguientes ejemplos.

```
alfax1 = "|NC";          alfax2 = #capunte1#alfax1#;  
alfax1 = "|C001NOMBRE"; alfax2 = #capunte1#alfax1#;  
alfax1 = "|C001TIPO";    alfax2 = #capunte1#alfax1#;  
alfax1 = "|C001LONGITUD"; alfax2 = #capunte1#alfax1#;
```

Los valores posibles, son los siguientes:

- "|NC" Con este valor se devuelve el número de campos que tiene el DEF (NOTA : Este valor cuenta también el campo 0)
- "|NP" Este valor devuelve el número de pantallas del DEF
- "|NK" Devuelve el número de claves del DEF.
- "|NR" Devuelve el número de relaciones del DEF.
- "|TITULO" Devuelve el título que aparece en la lista del menú principal del generador.
- "|FICHERO" Devuelve el path completo del fichero al que apunta el DEF.
- "|LONGITUD" Devuelve la longitud total del fichero .DAT
- "|TIPODATOS" Devuelve el tipo de datos DAT,DBF etc.
- "|DEF" Devuelve el path completo del nombre del DEF.
- "|IPFI" Devuelve el número interno.
- "|ID" Devuelve el número de versión.

También se pueden obtener datos de los campos de un DEF. Para esto la variable que pasaremos al DEF en vez del número de campo será una cadena que debe comenzar por "|CXXX". Donde con XXX indicamos el numero de campo del que se quiere extraer información, tras esta cadena indicaremos alguno de los siguientes parámetros :

- "LONGITUD" Devolverá la longitud del campo indicado
- "IOFFSET" Devolverá la posición que ocupa el campo físicamente dentro del .DAT (tomando como comienzo la posición de inicio del registro)
- "ILON" Devuelve la longitud del campo
- "ITIPO" Devolverá :
  - "F" si el campo es de tipo Fecha.
  - "A" si el campo es Alfanumérico.
  - "N" si el campo es Numérico.
  - "T" si el campo es de tipo Texto.

- "NOMBRE" Devuelve el nombre del campo.
- "IDCAMPO" Devuelve el contenido del apartado "Identificativo" de las características del campo.
- "MAXIMO" Devuelve el contenido del apartado "Máximo" de las características del campo.
- "MINIMO" Devuelve el contenido del apartado "Mínimo" de las características del campo.
- "MENSAJE" Devuelve el mensaje asociado a un campo.
- "DEFECTO" Devuelve el valor por defecto que tiene el campo

Otras utilidades:

- "|KXXX" Devuelve los campos que componen la clave XXX en el formato: NNAAABBBCCC ... donde:  
     NNN = numero de campos de la clave.  
     AAA, BBB..= los campos que componen la clave (desde 0)

## USO DE OCX.

Para usar los métodos de un OCX hay que tener en cuenta varias cosas:

Antes de nada, debemos asegurarnos que tenemos registrado el OCX en el registro de windows

(Para esto usamos el comando de consola : 'REGSRV32 /s PEPE.DLL' siendo PEPE.DLL el nombre del OCX)

Sabiendo esto, comenzaríamos :

1) Hay que activar el OCX (al igual que se ha de cargar una DLL para poder usarla) usando la sentencia:

```
|ACTIVA_OCX "nombre de la dll.nombre de la clase",Handle;
```

El retorno de esta instrucción es el handle del OCX en FSalida.

NOTA : Hay que recoger el handle de FSalida obligatoriamente, porque aunque se le indica en la

propia instrucción, debemos recogerla de allí.

2) Una vez activado el OCX, para pasarle parámetros lo hacemos de la siguiente forma:

- Cada parámetro se compone de dos variables : la primera que indica el tipo (0 es NULL, 1 es string,

...) y la segunda indica el valor que contiene el parámetro. Estas dos variables van encabezadas por

el parámetro de retorno (tipo y valor) que será el que se tome como referencia para el resto de

parámetros, es decir, en la instrucción de llamada daremos como referencia la variable de retorno

(por lo cual deberá existir obligatoriamente, aunque no haya retorno, en cuyo caso le daremos un tipo

NULL a dicho retorno). A partir de esta variable (sin contar esta, por supuesto) el programa mirará

cuantos parámetros necesita (supongamos n parámetros) y lee las 2\*n siguientes variables declaradas

(para cada parámetro tomara primero la variable de tipo y como segunda variable) en el mismo orden en

el que están declaradas en la sección del calculo 'VARIABLES'.

3) Cuando ya tengamos los parámetros establecidos, hacemos la llamada al OCX con la sentencia :

```
|OCX_INVOCA Handle, "Nombre del método", nume1, nume2,  
NumPar, PunteroParametros
```

Donde:

Handle es el handle del OCX obtenido en el punto nº 1.

Nombre del método es el nombre del método a invocar (debe estar entre comillas).

nume1, nume2 son dos numéricos que indican dos posiciones en pantalla, por si el método lo queremos 'acotar' solo a una parte de la pantalla.

NumPar es un numérico que indica el numero de parámetros que se pasan al método (incluido el de retorno).

PunteroParametros es un puntero que apunta a la variable que contiene el tipo del retorno.

4) Hecho esto y si no vamos a usar más el OCX, debemos descargarlo de memoria (al igual que una DLL). Esto se hace con la instrucción:

```
|OCX_LIBERA Handle
```

Donde handle es el handle del OCX.

Veamos un ejemplo :

Tenemos un OCX llamado 'NOMBRES.DLL'. Dentro de este OCX tenemos una clase llamada 'ManejaNombres'

y un método (el que vamos a usar) llamado 'PintaNombres' el cual tiene tres parámetros de tipo

string : Nombre, primer\_apellido y segundo\_apellido.

Lo primero que debemos tener registrado es el OCX (Desde una tarea de interfaz de comandos escri-

biríamos:

```
REGSVR32 /s NOMBRES.DLL
```

Preparamos en la declaración de variables, los parámetros que vamos a pasar al método del OCX.

```
|VARIABLES;
```

```
.
```

```
.
```

```
HandleOCX = -1;
```

```
Puntero = 0;
```

```
.
```

```
.
```

```
TipoRetorno = 0; (Tipo NULL, ya que no lo necesitamos)
```



```

ValorRetorno = -1,

TipoNombre = 1; ValorNombre = ""; (Tipo String)

TipoApellido1 = 1; ValorApellido1 = ""; (Tipo String)

TipoApellido2 = 1; ValorApellido2 = ""; (Tipo String)

.

.

|FIN;

```

Hecho esto, cargaríamos (ya desde el calculo) el OCX en memoria :

```

|ACTIVA_OCX "NOMBRES.ManejaNombres",HandleOCX;

|SI FSalida ] 0;

HandleOCX = FSalida;

|SINO;

|ACABA;

|FINSI;

```

Y ahora daríamos valores a los parámetros :

```

ValorNombre = "Pepe";

ValorApellido1 = "Martinez";

ValorApellido2 = "Lopez";

Puntero = TipoRetorno<-;

```

Hacemos la llamada al método ....

```

|OCX_INVOCA HandleOCX, "PintaNombres", -1, -1, 4,
Puntero;

```

... y una vez que hemos acabado, liberamos la memoria cerrando el OCX

```
|OCX_LIBERA HandleOCX;
```

## MANEJO DE ARCHIVOS EXCEL.

Ejemplo de manejo:

```
aAbre = "C:\CAJON\XDOCUMENTOS\OTP\XEVALUACION.XLS";
```

```
|CARGADLL "agixl97.dll";    ||Cargamos la dll
```

```
|SI FSalida < 0;
```

```
    |MENAV "0000No se puede usar agixl97.dll";
```

```
    |ACABA;
```

```
|FINSI;
```

```
|ALFACALLDLL "agixl97.dll", "carga_book", aAbre;    ||Abrimos el  
libro xevaluacion.xls
```

```
|ALFACALLDLL "agixl97.dll", "selecciona_hoja", "Evaluacion";  
||Seleccionamos la hoja Evaluación del libro xevaluacion.xls
```

```
|PARA; |SI; |HACIENDO;
```

```
    aCampoA = "A" + nLinea;
```

```
    |ALFACALLDLL "agixl97.dll", "peek_dato", aCampoA;  
    ||Lee el dato de la celda aCampoA (p.ej. A1) y lo guarda en
```

```
        ||aCa  
        mpoA
```

```
    |QBF aCampoA;
```

```
    aCampoB = "B" + nLinea;
```

```
    |ALFACALLDLL "agixl97.dll", "peek_dato", aCampoB;
```

```
    |QBF aCampoB;
```

```
    nLinea = nLinea + 1;
```

```
|SI nLinea > 1024; |SAL; |FINSI;
```

[SIGUE;](#)

aComillas = A”;

aAlfa = “A1” +”,”+ aComillas +“PEPE”+ aComillas;

|ALFACALLDLL "agixl97.dll", "poke\_dato", aAlfa;    ||Introducimos  
“Pepe” en la celda “A1”

|ALFACALLDLL       "agixl97.dll",       "fin\_book",       "imprime";  
||Imprimimos

|ALFACALLDLL "agixl97.dll", "fin\_book", "salva";    ||Guardamos

|ALFACALLDLL "agixl97.dll", "fin\_book", "";        ||Cerramos el libro  
excel

[DESCARGADLL](#) "agixl97.dll";    ||Descargamos la librería

## **6.- Normas para el bloqueo de registros**

Nota: El bloqueo se utiliza para que temporalmente seamos propietarios de un determinado registro y nadie más nos interfiera en el alta, baja o modificación del mismo y, además, para que no interfiramos en el alta, baja o modificación de un registro de otro usuario.

- ¿Cuándo hace falta bloquear un registro? ¿Al leer, al grabar o al borrar?
- Lectura: cuando vayamos a regrabarlo o a borrarlo.
- Grabación: cuando se trata de una pregrabación para una posterior regrabación.
- En caso de bloquear, ¿cuándo debemos utilizar el flag de espera?
- El flag de espera se activará cuando queramos que el programa detecte por sí solo la liberación del registro que intentamos bloquear.
- ¿Qué flag de mensajes sería el correcto en cada caso?

- Utilizar el '4' solamente cuando el control de las interferencias por bloqueos se realizan manualmente.

- Utilizar el '0' normalmente, solamente se genera mensaje en caso de intentar bloquear un registro bloqueado.

- Utilizar el '2' solamente cuando queramos que se generen todos los mensajes de error. En la grabación y borrado utilizar siempre éste. [|GRABA 020#??](#). Las grabaciones y los borrados siempre controlan si el registro está o no está, por lo tanto cualquier posible error debemos conocerlo.

- Sintaxis de los bloqueos en [|LEE](#), [|DEFBUCLE](#) y [|BUCLELIN](#).

[|LEE](#) abc#1= donde:

a → flag de bloqueo:      0 → sin bloqueo

1 → con bloqueo

b → flag de mensajes:      2 → todos los mensajes

4 → ningún mensaje

0 → solamente en caso de

bloqueo

c → flag de espera:      0 → no se espera si está  
bloqueado, da el mensaje y un  
'Pulsa Tecla'.

[|DEFBUCLE](#) agifa024;

#1#1;

;

"000000";

"999999";

**ab**;

NULL, agifa024\_1;

[|FIN](#); donde:

a → flag de bloqueo

b → bloquea los registros

b → flag de espera:

e → espera hasta que se libere

|[BUCLELIN](#) aLineas#1; donde:

a →

0 → bloqueando los registros

1 → borrando las líneas  
(¡CUIDADO!)

2 → sin bloqueo

Ejemplos:

A) Regrabar un registro que ya existe.

|[DDEFECTO](#) #1;

|[ABRE](#) #1;

#1#0 = "REGRABAR";

|[LEE](#) 101#1=;

|[SI](#) FSalida=0;

#1#1="HOLA";

|[GRABA](#) 020#1a;

|[FINSI](#);

|[CIERRA](#) #1;

B) Borrar un registro.

|[ABRE](#) #1;

#1#0 = "BORRAR";

|[LEE](#) 101#1=;

|[SI](#) FSalida=0;

|[BORRA](#) 020#1a;

|[FINSI](#);

|[CIERRA](#) #1;

C) Pregrabar un registro en blanco para su posterior relleno y liberación.

```
|DDEFECTO #1;  
|ABRE #1;  
#1#0 = "PREGRABAR";  
|LEE 101#1=;  
|SI FSalida ! 0;  
    |GRABA 020#1b;  
|FINSI;  
|ENDATOS #1#1;  
|SI FSalida=0;  
    |GRABA 020#1a;  
|FINSI;  
|LIBERA #1;  
|CIERRA #1;
```

D) Grabar si no existe, regrabar si existe.

```
|DDEFECTO #1;  
|ABRE #1;  
#1#0 = "GRABAR-REGRABAR";  
|LEE 101#1=;  
nSalida=FSalida;  
#1#1="HOLA";  
#1#2="PEPE";  
|SI nSalida ! 0;  
    |GRABA 020#1n;  
|SINO;
```

|[GRABA](#) 020#1a;

|[FINSL](#);

|[CIERRA](#) #1;

## 7.- Prácticas

### 7.1.- Tablas únicas.

1.- Crear una tabla de Provincias que se llamará *ttma0001* con los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	Pr_Codigo	Alfanumérico	2
1	Pr_Nombre	Alfanumérico	20
2	Pr_Prefijo	Alfanumérico	3

Añadirle 2 índices:

1.- Campo 0

2.- Campo 1.

Crear su pantalla con el nombre *ttma0001.pan*.

Compilar.

Anular colores.

Probar.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                      {{ttma0001.tab}}

2.- Crear una tabla de Poblaciones que se llamará *ttma0002* con los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	Po_Codigo_Pro	Alfanumérico	2
1	Po_Codigo_Pob	Alfanumérico	3



2	Po_Nombre_Pob	Alfanumérico	30
---	---------------	--------------	----

Añadirle 2 índices:

1.- Campo 0 y 1.

2.- Campo 2.

Crear su pantalla con el nombre *ttma0002.pan*.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                      {{ttma0001.tab}}

10,20:Poblaciones                      {{ttma0002.tab}}

Compilar.

Anular colores.

Probar.

3.- Crear una tabla de clientes que se llamará *ttma0003* con los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	CI_Codigo	Numérico	6
1	CI_Nombre	Alfanumérico	40
2	CI_Dirección	Alfanumérico	40
3	CI_CP1	Alfanumérico	2
4	CI_CP2	Alfanumérico	3
5	CI_Telefono	Alfanumérico	15
6	CI_Fax	Alfanumérico	15

7	CI_Nif	Alfanumérico	15
8	CI_RE	Alfanumérico	1

Añadirle 4 índices:

- 1.- Campo 0.
- 2.- Campo 1.
- 3.- Campo 5.
- 4.- Campo 7.

Valores posibles en campo 8: S N s n

Crear su pantalla con el nombre *ttma0003.pan*.

Compilar.

Anular colores.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                    {{ttma0001.tab}}

10,20:Poblaciones                    {{ttma0002.tab}}

10,30:Clientes                        {{ttma0003.tab}}

Probar.

4.- Crear una tabla de Artículos que se llamará *ttma0006* con los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	Ar_Codigo	Alfanumérico	20
1	Ar_Descripción	Alfanumérico	40

2	Ar_Precio	Numérico	10
3	Ar_Tipolva	Numérico	1

Añadirle 1 índices:

1.- Campo 0

Crear su pantalla con el nombre *ttma0006.pan*.

Compilar.

Anular colores.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                    {{ttma0001.tab}}

10,20:Poblaciones                {{ttma0002.tab}}

10,30:Clientes                    {{ttma0003.tab}}

10,40:Artículos                    {{ttma0006.tab}}

Probar.

5.- Crear una tabla de Tipos de I.V.A. que se llamará *ttma0007* con los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	TI_Codigo	Numérico	2
1	TI_Descripción	Alfanumérico	40
2	TI_PIVA	Numérico	2
3	TI_PRE	Numérico	2

Añadirle 1 índices:

### 1.- Campo 0

Crear su pantalla con el nombre ttma0007.pan.

Compilar.

Anular colores.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                    {{ttma0001.tab}}

10,20:Poblaciones                {{ttma0002.tab}}

10,30:Clientes                    {{ttma0003.tab}}

10,40:Artículos                    {{ttma0006.tab}}

10,50:Tipos de IVA                {{ttma0007.tab}}

Probar.

### 7.2.- Tablas Cabecera-líneas.

1.- Crear una tabla maestra de facturas que se llamará *ttma0004* con los siguientes campos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	CFr_Numero	Numérico	6
1	CFr_Fecha	Fecha	40
2	CFr_Cliente	Numérico	6
3	CFr_NomCliente	Alfanumérico	40
4	CFr_Dto_PP	Numérico	2
5	CFr_Bruto	Numérico	10

6	CFr_Base	Numérico	10
7	CFr_Por_IVA	Numérico	2
8	CFr_TT_IVA	Numérico	10
9	CFr_Por_RE	Numérico	2
10	CFr_TT_RE	Numérico	10
11	CFr_TT_Factura	Numérico	10

Añadirle 2 índices:

- 1.- Campo 0.
- 2.- Campo 2.

Crear su pantalla con el nombre *ttma0004.pan*.

- 2.- Crear una tabla detalle de facturas que se llamará *ttma0005* con los siguientes campos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	LFr_Numero	Numérico	6
1	LFr_Linea	Numérico	3
2	LFr_Articulo	Alfanumérico	20
3	LFr_DesArticulo	Alfanumérico	40
4	LFr_Unidades	Numérico	4
5	LFr_Precio	Numérico	10
6	LFr_Por_Dto	Numérico	2

7	LFr_TT_Dto	Numérico	10
8	LFr_TT_Linea	Numérico	10

Añadirle 2 índices:

- 1.- Campo 0 + Campo 1.
- 2.- Campo 2.

La tabla *ttma0005* es líneas de la tabla *ttma0004*. El campo lineal es el 1.

Crear su pantalla usando la de cabeceras que es la *ttma0004.pan*.

Compilar.

Anular colores.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias                    {{ttma0001.tab}}

10,20:Poblaciones                {{ttma0002.tab}}

10,30:Clientes                    {{ttma0003.tab}}

10,40:Artículos                   {{ttma0006.tab}}

10,50:Tipos de IVA               {{ttma0007.tab}}

20:Ventas

20,10:Facturas                   {{ttma0004.tab}}

Probar.

### 7.3.- Tablas relacionadas.

1.- Añadir una relación de la tabla *ttma0002* en su campo 0 respecto a la tabla *ttma0001*.

2.- Añadir una relación de la tabla *ttma0003* en sus campos 3 y 4 respecto a la tabla *ttma0002*. Que al escoger un dato de la tabla relacionada se pinten sus valores descriptivos en pantalla.

3.- Añadir una relación de la tabla *ttma0004* en su campo 2 respecto a la tabla *ttma0003*. Poner el campo 3 como no modificable y que su valor por defecto sea el campo 1 de la tabla *ttma0003*. Poner el campo 4 como no modificable y que su valor por defecto sea el campo 8 de la tabla *ttma0003*.

4.- Añadir una relación de la tabla *ttma0005* en su campo 2 respecto a la tabla *ttma0006*. Poner los campos 3 y 5 de la tabla *ttma0005* como no modificables. El valor por defecto del campo 3 será el campo 1 de la tabla *ttma0006*. El valor por defecto del campo 5 será el campo 2 de la tabla *ttma0006*.

Compilar.

Probar.

#### **7.4.- Cálculos y Procesos.**

1.- Agregar un cálculo o proceso en el campo 5 del def *ttma0003* que se llame PonPrefijo y se ejecute antes de entrar en el campo y que ponga el prefijo telefónico y pulse 3 veces la flecha hacia la derecha, teniendo en cuenta que el cálculo sólo deberá efectuarse en caso de que se entre en la ficha en modo alta.

Compilar.

Probar.

2.- Agregar un cálculo en el campo 7 del def *ttma0003* que se llame CalculaLetra y se ejecute al salir del campo. Este proceso deberá calcular la letra del N.I.F., teniendo en cuenta que el cálculo sólo deberá efectuarse en caso de que el usuario pulse la tecla F12. Tener en cuenta que el cálculo de la letra en NIF es el siguiente:

- Se divide el NIF en estado numérico entre 23 y se coge su resto, este se busca en una tabla para comprobar la letra. La tabla es la siguiente:

1	R
2	W
3	A
4	G
5	M
6	Y
7	F
8	P
9	D
10	X
11	B
12	N
13	J
14	Z
15	S
16	Q
17	V
18	H
19	L
20	C
21	K
22	E



Compilar.

Probar.

3.- Agregar un cálculo en el campo 6 del def *ttma0005* cuya función sea actualizar el valor del campo 8 del mismo fichero con una operación que será:

$$\text{campo\_8} = \text{unidades} * \text{precio} - \text{dto}$$

Compilar.

Probar.

4.- Agregar un cálculo en el campo 8 del def *ttma0005* que sea de tipo reversible y cuya función será actualizar los campos de la cabecera de la factura. Atención al acumulador de R.E.; éste deberá actualizarse en función del campo 8 de la ficha del cliente.

Compilar.

Probar.

(...) Hacer punto 6.6.

5.- Hacer un cálculo en el campo 0 del def *ttma0003* que busque en el fichero de configuración, ponga el código de cliente automáticamente y después lo incremente en la ficha de configuración.

6.- Hacer un cálculo en el campo 0 del def *ttma0004* que busque en el fichero de configuración, ponga el número de factura automáticamente y después lo incremente en la ficha de configuración.

7.- Hacer un cálculo en el campo 3 del def *ttma0006* que busque en el fichero de configuración, que se ejecute antes de entrar en el campo y ponga el tipo de IVA automáticamente.

## 7.5.- Rutinas.

1.- Hacer una rutina que abra el maestro de tipos de IVA y lo posicione en un registro. Ejecutar esta rutina desde el proceso reversible que calcula los acumulados en la cabecera.

### 7.6.- Programas.

1.- Crear un mantenimiento de un fichero de configuración que se llamará *ttma9999* y tendrá los siguientes datos:

CAMPO	NOMBRE	TIPO	TAMAÑO
0	CO_Clave	A	1
1	CO_Facturas	Numérico	6
2	CO_Clientes	Numérico	6
3	CO_IVADef	Numérico	1

Añadirle 1 índices:

1.- Campo 0

Crear su pantalla con el nombre *ttma9999.pan*.

Compilar.

Anular colores.

Añadir la opción en el menú:

10:Maestros

10,10:Provincias	{{ttma0001.tab}}
10,20:Poblaciones	{{ttma0002.tab}}
10,30:Clientes	{{ttma0003.tab}}
10,40:Artículos	{{ttma0006.tab}}
10,50:Tipos de IVA	{{ttma0007.tab}}
10,60:Configuración	{{ttma9999.tab}}

Crear un mantenimiento para este fichero de manera que:

- Al entrar, si no existe la ficha, se cree automáticamente.
- Al salir, se grabe la ficha.
- No muestre las opciones ALTA-BAJA-CONSULTA-MODIFICACIÓN, sino que entre directamente en la ficha (sólo debe haber 1).

Probar.

### **7.7.- Bucles.**

1.- Hacer un bucle que nos muestre por pantalla el acumulado de las unidades vendidas en las líneas de factura.

2.- Hacer un bucle que nos permita borrar los artículos existentes pidiendo código inicial y código final.

3.- Hacer un bucle que, pidiendo en sus márgenes *desde y hasta* número de factura, fecha y cliente, borre las facturas que estén dentro de ese margen (y consecuentemente, en cada cabecera de factura que borre, que borre sus líneas).

4.- Hacer un proceso de tipo de impresión de manera que, cuando estemos consultando una factura en el proceso de facturas, nos permita imprimirla. Para ello utilizaremos una impresión de la cabecera y un bucle para la impresión de las líneas.

## **8.- Guía de Errores.**

### **8.1.- Lista de avisos del Generador.**

1 =

2 =

3 =

4 =

5 = Error interno en [MENUG](#)

- 6 =
- 7 = ERROR al abrir MDAUDIT.LOG
- 8 = Clave de mas de 150 caracteres
- 9 = Error en Tratamiento campos MEMO
- 10 = Error en Tratamiento campos MEMO
- 11 = Error en Tratamiento campos MEMO
- 12 = Error en Tratamiento campos MEMO
- 13 = Error en Tratamiento campos MEMO
- 14 = Error en Tratamiento campos MEMO
- 15 = Error en Tratamiento campos MEMO
- 16 = Error en Tratamiento campos MEMO
- 17 = Error en Tratamiento campos MEMO
- 18 = Error en Tratamiento campos MEMO
- 19 = Error en Tratamiento campos MEMO
- 20 = Error en tratamiento MEMO desplazamiento
- 21 = Error en tratamiento MEMO desplazamiento
- 22 = Error Interno Calculadora
- 23 = Error Interno Calculadora
- 24 = Error Interno Calculadora
- 25 = rmdrun.usr Inaccesible
- 26 = No se puede abrir el MDAUDIT.LOG
- 27 = No ha lugar h000
- 28 = Exceso de variables en INFORME
- 29 = Calculo ilegal en INFORME

30 = Exceso de operaciones en INFORME

31 = Exceso de condiciones en INFORME

32 = Condicion ilegal en INFORME

33 = Calculo ilegal en INFORME

34 = Informe ilegal en INFORME

35 = Fichero ilegal en INFORME

36 = Informe ilegal en INFORME

37 = Variable ilegal en INFORME

38 = Informe ilegal

39 = Informe ilegal

40 = Error Interno en memo

41 = Error Interno en memo

42 = Error de CORRUPCION o VERSION ANTERIOR

43 = Error en Puntero, (tipo ilegal de variable)

44 = Error en Activacion de Alarma

45 =

46 = Numero de Clave Erroneo

47 = Numero de Campo Erroneo

48 = Error variable > 32767 o < -32767

49 =

50 = Linea 0 en fichero de LINEAS (LECTURA?)

51 = Linea 0 en fichero de LINEAS (GRABACION?)

52 = Sobrepasado el numero maximo de defs (10)

53 = Error al posicionar un campo (TRATAMIENTO DE PANTALLAS)

54 = NO ES ERROR

55 = Línea BLOQUEADA

56 = ERROR asignación de memoria

57 = ERROR asignación de memoria

58 =

59 = ERROR en bloques campo MEMO

60 = Nombre de DEF existente

61 =

62 = Nombre DEFBUCLE no encontrado

63 = Nombre DEFBUCLE no encontrado

64 = Demasiados PROCESOS

65 = Nombre no encontrado (PROCESOS???)

66 =

67 =

68 =

69 =

70 =

71 =

72 =

73 =

74 =

75 =

76 =

77 = Error en Actualización RELACIÓN

78 =

79 = El campo no es CLAVE

## 8.2.- ERRORES DE TRATAMIENTO DE FICHEROS C-ISAM.

Números	Descripción
---------	-------------

=====

=====

100		Grabando registro nuevo ya existente. Duplicado.
101		Fichero no abierto.
102	(*)	Uno de los argumentos tiene un valor con un significado no definido.
103	(*)	Los valores de la clave no son válidos.
104	(*)	No se pueden abrir más ficheros.
105	(*)	Fichero corrupto.
106	(*)	Acceso exclusivo al fichero no posible.
107	(*)	Registro bloqueado para sólo lectura por otro proceso.
108	(*)	El valor de la clave ya ha sido establecido como clave.
109	(*)	La función ejecutada no puede ser referida a la clave primaria.
110	(*)	Principio o fin de fichero.
111		Registro no encontrado.
112	(*)	Regrabando un registro sin haberlo leído anteriormente.
113	(*)	Fichero bloqueado.
114	(*)	Nombre de fichero demasiado largo.
115		No se pueden bloquear más registros.

116 (\*) No hay memoria para proceder.

### 8.3.- Errores no usuales en instalaciones. Errores de desarrollo.

Números Descripción

=====

=====

100 La grabación de registros nuevos está controlada desde programa, no obstante puede escaparse alguna combinación en la que se provoque un error. Dejando esta opción como límite, lo normal en estos casos es que el fichero en cuestión este dañado por lo que se debería realizar una 'Creación de Índice' y repetir el proceso. Cuando se genera este error los registros afectados no serán grabados.

101 El que no esté abierto un fichero cuando se está operando sobre él suele ser causa de limitación de ficheros abiertos en la configuración del sistema. Si se trata de D.O.S. habría que revisar el CONFIG.SYS, si en cambio es UNIX habría que hacer referencia a los permisos que se atribuyen al fichero. En segundo lugar, si nos sigue fallando, tendríamos que hacer lo contrario.

111 Simplemente es que el registro no se encuentra. Forma parte del mensaje que da el entorno en las relaciones que no permiten el alta y la existencia del registro es obligada. En la ejecución de programas es posible que se genere cuando no se encuentra un registro necesario para seguir con la misma. Por ejemplo, si estamos calculando nóminas y no existe la empresa.

115 Este error normalmente es causa de un error de programación. Cuando se están actualizando registros estos



se van bloqueando uno a uno, si no se desbloquean el C-ISAM soporta un tope de 10 ó 20, a partir de ahí se generará el error por cada registro.