

# Objetos Natural (I)

La aparición de NATURAL en 1985 con su versión 1, supuso, al menos en España, una revolución en el mundo de los Grandes Sistemas, siendo superada tres años mas tarde con la versión 2. El elemento diferenciador de este producto era un lenguaje de 4ª generación, que, entre otras muchas cosas favorecía la Modularidad.

## 1. Introducción

No hace mucho, un amigo del autor de este artículo le contaba que había sido contratado por una empresa para mejorar la gestión de un almacén que fabrica 2000 puertas al día, lo cual, como es lógico, exige un cierto orden a la hora de gestionar los stocks para así optimizar el proceso de fabricación. Para poder llevar a cabo su misión, tuvo que empezar por convencer al encargado del almacén de que su cometido consistía en enseñarle a hacer las cosas de una nueva forma, pero sobre todo debió superar los recelos de éste, y el comentario siguiente: *'Esto lo he llevado yo 20 años, ahora no va a venir nadie a decirme como lo debo hacer'* a lo que mi amigo le contesto que una persona puede estar haciendo las cosas mal desde el principio, 1 , 20 y 60 años, sin que eso sea justificación de que las cosas se hacen bien.

Este, que es un hecho real, también aparece, y muy a menudo, en el ámbito de los entornos de desarrollo informáticos, pues se debe tener presente que en estos entornos no llega al **5%** el personal que cuenta con titulación informática universitaria<sup>1</sup>, por lo que la gran mayoría de los desarrolladores se han autoformado, muchos de ellos sin poder preguntar a expertos ya que fueron enviados a su puesto de trabajo como *tales* por una empresa de servicio, y, con el tiempo crearon su propia *'escuela'*.

Este artículo se marca como objetivo el ayudar a los programadores de un entorno de desarrollo NATURAL a utilizar cada uno de los tipos de objetos de que dispone esta plataforma, para lo cual, el planteamiento que se va a seguir es definir cada uno de los tipos existentes y comentar los errores mas frecuentes, pero, no se va a explicar ni la sintaxis ni los comandos de los editores usados para crear cada uno de ellos, ya que esto alargaría innecesariamente el artículo.

Todas las observaciones que se hagan en el presente artículo estarán referidas a la versión 2 de Natural, y mas concretamente a NATURAL en modo estructurado, salvo indicación expresa en otro sentido.

---

<sup>1</sup>En este porcentaje, se incluye tanto al personal propio como al subcontratado, reduciéndose dicho porcentaje en los niveles de dirección de proyectos y superiores. Y, por lo que se refiere a la titulación, hay que tener en cuenta que, debido a la gran demanda de finales de los 80, muchos son los que dicen tener, pero pocos son los que realmente tienen.

## 2. Fuentes y Objetos

La arquitectura de NATURAL organiza, de cara al usuario, los programas en librerías, conteniendo cada una de ellas todos los módulos una aplicación<sup>2</sup>. Y, para todos aquellos módulos que son de uso común de la instalación, cuenta con una librería en la que, por defecto, el sistema buscara el modulo que no encuentre en la librería de su aplicación. Esta suele ser la librería **SYSTEM**, aunque puede tener cualquier otro nombre ya que este es asignado por el administrador de la instalación.

Dentro del entorno NATURAL, los continentes de los códigos fuentes y de los códigos ejecutables tienen el mismo nombre o identificador, siendo esto posible porque sus contenidos se almacenan en lugares distintos, gestionando el propio sistema este almacenamiento.

Los ficheros ADABAS<sup>3</sup> relacionados con esta función son:

**FNAT** : contiene los programas del sistema y de las utilidades.

**FUSER** : Contiene las librerías de aplicaciones FUENTES y OBJETOS.

**FDIC** : Contiene los Módulos de Definición de Datos (DDM).

Así, en un entorno NATURAL, se entiende por **FUENTE**, el código fuente escrito por el programador; y **OBJETO**, el código resultante de la compilación del fuente.

Los comandos del sistema que permiten manejar estos continentes cuando se tiene un fuente en el buffer del editor, son:

**SAVE**: el código fuente se guarda en el fichero del sistema.

**CAT**: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se almacena en el fichero del sistema. No se guarda el fuente.

**STOW**: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se almacena en el fichero del sistema, y además, en este caso, también se almacena el fuente. Equivale por tanto a **SAVE + CAT**.

Los comandos contrarios de estos son:

**PURGE**: Borra un fuente del fichero del sistema.

**UNCAT**: Borrar un objeto del fichero del sistema.

**SCRATCH** : Borrar fuente y objeto.

**RUN**: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se ejecuta. No se guarda el fuente ni el objeto.

---

<sup>2</sup>Por esta razón, las **librerías** de Natural también se denominan **aplicaciones**.

<sup>3</sup>Los **fuentes**, dependiendo del sistema, pueden guardarse en un fichero ADABAS o como ficheros secuenciales del sistema operativo.

### 3. Tipos de Objetos

La gran ventaja proporcionada por NATURAL en sus comienzos, junto a la de ser un lenguaje de 4ª generación, era la modularización que ofrecía, gracias a todos los tipos de objetos relacionados en la **figura 1**. Esto facilitaba la labor de programación, pues cambiaba los interminables programas COBOL por funciones elementales, capaces de resolver el mismo problema pero de un forma más simple y con menos código.

Grupo	Carácter asociado	Tipo
Áreas de Datos	G	- Global
	A	- Parameter
	L	- Local
Proceso	P	- Programas
	N	- Subprogramas
	S	- Subrutinas externas
Sistemas de ayuda	H	- Helptext
	H	- Helproutines
Otros	C	- Copycodes
	M	- Mapas
	T	- Texto
	D	- Descripción
	R	- Recording
DDM	V	- Modulo Definición Datos

**Figura 1**

La **Figura 1** muestra los distintos tipos de módulos, también llamados *objetos*, clasificados por grupos, así como el carácter con el que generalmente se asocia el tipo. Este carácter suele ir, dependiendo de la nomenclatura asumida por la instalación, en una posición predeterminada, generalmente al final del nombre. Así, el nombre JMP0000**G** representa un área de datos Global, mientras que el JMPXUT1**P** representa el nombre de un objeto tipo programa.

Así mismo, se ha incluido el tipo de objeto **DDM** por ser muy usado por todos los programadores, aunque solo pueda ser definible y modificable por el DBA<sup>4</sup>. Este tipo de objeto, ya tratado en el artículo dedicado a ADABAS, entre otras cosas, se diferencia de los que se tratan aquí en que la DDM solo define las características de los campos de un archivo o fichero de la Base de datos, es decir, no reserva memoria para ningún buffer, cosa que si hacen los distintos objetos agrupados bajo el título **ÁREA DE DATOS**, los cuales se basan en las características definidas en las DDMs para reservar el espacio correspondiente al buffer que servirá de intercambio de información con ADABAS. Otra diferencia se encuentra en el nombre, el cual puede tener hasta 32 caracteres, dando esto significado a su contenido.

<sup>4</sup>DBA : Data Base Administrator o Administrador de la Base de Datos.

## 4. Características de los tipos

### 4.1 ÁREAS DE DATOS

Como el lector sabe muy bien, un **programa ejecutable**, con independencia del lenguaje en el que se haya escrito el fuente, no es más que la unión de los siguientes conjuntos:

- uno de posiciones de memoria, a las que se les asigna un nombre lógico para su más fácil manejo por el programador. Son las variables; y
- otro con la secuencia de instrucciones que debe ejecutar el procesador para manejar la información contenida en las posiciones de memoria definidas en el primer conjunto.

Pues bien, en el modo estructurado de Natural, cualquier objeto de tipo proceso debe especificar al principio del mismo las variables y los buffers que se van a usar en el programa, al igual que ocurría con la **Data División** de COBOL. Estas variables, atendiendo al uso que se va a hacer de su contenido, pueden ser de dos tipos:

- **Uso local** : su contenido solo puede ser utilizado por el módulo en el que se encuentra definida la variable y no se comparte su valor por los distintos ejecutables que contienen su definición.
- **Uso Global**: Su contenido es compartido por todos los programas de un usuario que contengan su definición y que se encuentren en la misma librería, de modo que una variable, puede cargar su valor en un programa para ser leído su contenido en otro.

Así pues, la definición de las variables locales, y/o de las estructuras se debe hacer dentro del programa y concretamente, al principio del mismo, tal y como muestra la **figura 2**, utilizando la sentencia **DEFINE DATA**. Pero, puede darse el caso de que un mismo conjunto de variables tenga que definirse en varios módulos ejecutables. Para evitar el tener que escribir en cada uno de esos módulos todas las definiciones, se puede usar el tipo de objeto **ÁREA de DATOS**, el cual no es un objeto ejecutable en el amplio sentido de la palabra, pero sus definiciones se incorporan al ejecutable en el momento de la compilación, con lo que se reduce el número de líneas de fuente, tal y como se comprueba en el ejemplo de la **figura 2**.

```
EDIT-NAT:UTIL-JMP (LISOBJOH)-Helproutine->Struct-Free-0 ----- Columns 001 072
COMMAND==> SCROLL==> CSR
***** ***** top of data *****
000010 *****
000020 *
000040 *          PROGRAMA : LISOBJOH
000060 *          AUTOR   : JMP
000070 *
000080 *****
000090 DEFINE DATA
000100 GLOBAL USING UT000001 WITH PRINCIPAL.BLOQUE1
000110 PARAMETER
000120          1 W-COD-ERROR      (N4)          /* CODIGO ERROR
000130 LOCAL USING UT1-ERR1      /* AREA-PARAMETROS RETORNO-PARAM
000140 LOCAL
000150          1 W-TEXTO-ERROR (A59)
000170 END-DEFINE
000180 * -----
000190 COMPUTE NUMERO-ERROR = 2100 + W-COD-ERROR
000200 ASSIGN APLICACION = 'UTILIDAD'
000210 CALLNAT 'UT1ERR2N' AREA-PARAMETROS RETORNO-PARAMETROS
000220 COMPRESS TEXTO-CORTO W-COD-ERROR INTO W-TEXTO-ERROR
000230 SET CONTROL 'WFL69C1B1/3'
000240 INPUT USING MAP ' LISOBJOH'
000260 END
***** ***** bottom of data *****
```

Figura 2

Estas áreas de datos pueden ser de varios tipos, y, si bien la definición de las variables sigue el mismo mecanismo que si se definieran dentro del módulo, la diferencia fundamental entre ellas estriba en su disponibilidad en función del tipo de objeto que los utiliza.

- **Global** : contiene la definición de las variables globales, con la restricción de que solo puede usarse un área global por programa. Y, como particularidad, tiene el hecho de agrupar en bloques las variables que

contiene, de modo que solo se reserve espacio en el programa para aquellas variables contenidas en los bloques que se especifiquen.

- **Local** : contiene la definición de variables que serán usadas en modo local dentro del programa. No tienen bloques, y las definiciones pueden ser de variables independientes, estructuras (o variables redefinidas) y definiciones de buffers asociados a vistas de ficheros. **Figura 2.**

- **Parameter**: Este tipo de área solo puede ser usada en objetos del tipo subrutina y subprograma. Y contiene la definición de las variables que serán recogidas como parámetros iniciales en el programa o función llamado.

Un tipo especial de variable es aquella que referencia campos de un fichero. En este sentido, cabe recordar que Natural, es un lenguaje íntimamente relacionado con el SGDB **ADABAS**<sup>5</sup>. Los ficheros son definidos físicamente por el administrador de la Base de datos (**DBA**), pero son gestionados por ADABAS. Para que el programador pueda grabar o recuperar información de estos ficheros, el administrador, (que es una persona del departamento de sistemas<sup>6</sup>) pone a su disposición unos módulos denominados **DDM** (Definition Data Module). En ellos se encuentran definidos solo los nombres y atributos de aquellos campos del fichero que pueden ser vistos por el programador, con lo que se está restringiendo el acceso a datos confidenciales. Evidentemente, puede haber DDMs que contengan las definiciones de todos los campos de un fichero, y restringir el acceso a la información de determinados campos mediante el uso de **NATURAL SECURITY** (otra utilidad proporcionada por el sistema NATURAL que gestiona y controla las autorizaciones de acceso).

Pues bien, el buffer o posiciones de memoria que servirán de tampón intermediario para enviar y/o recibir datos de la base, también se debe encontrar definido dentro de cualquiera de los tipos de áreas especificados, aunque lo normal es que se defina dentro de un área local, y no dentro de una global, ni que se pase como parámetro a otro objeto.

Esta estructura cuenta con la siguiente particularidad : No necesita definir los atributos de formato y longitud de cada campo, pues asume los que se encuentren definidos en la DDM, aunque en caso de definirlos, esta afectaría solo a su longitud, no a su tipo; y además, no necesita definir todos los campos de la DDM, solo aquellos campos que se vayan a usar dentro del programa. Por esta razón se denomina **VISTA (view)**.

Entre las recomendaciones dadas por **SOFTWARE AG**<sup>7</sup>, se encuentra el aconsejar el uso de **view internas**, tal y como muestra la **figura 3** ya que esto reduce el tamaño del objeto, y además aumenta la eficiencia de **ADABAS** ya que este gestor debe generar un **IFB** (Internal Format Buffer) por cada campo de la vista, lo cual es muy costoso en consumo de CPU.

---

<sup>5</sup> Ver artículo dedicado al Sistema Gestor de Base de Datos (SGDB) **ADABAS** en **SOLO PROGRAMADORES** de Diciembre de 1995 num. 16.

<sup>6</sup> Como anécdota el autor recuerda que en una determinada instalación, el DBA tenía prohibido el uso de la sentencia REINPUT por no ajustarse esta sentencia estrictamente a los principios de la programación estructurada enunciados por **Dijkstra**.

<sup>7</sup> **Software AG** es la empresa alemana que diseñó y creó **ADABAS** y **NATURAL**. En España este producto es distribuido por **SOFTWARE AG ESPAÑA**.

```

UTIL-JMP JMPDES          Utilidad para desarrollo          16/04/96 15:17:25
                          Libreria: UTIL-JMP Objeto: LISOBJOB
-----
0090 DEFINE DATA
0100     GLOBAL USING UT000001 WITH PRINCIPAL.BLOQUE1
0110     LOCAL
0140     1 FUSER-D VIEW OF SYSTEM-FUSER
0150     2 SRCID
0160     2 REDEFINE SRCID
0170     3 LIBERIA      (A8)
0180     3 PROGRAMA    (A8)
0190     3 REGISTRO    (B2)
0200     2 C*SRCTX
0210     2 SRCTX       (1:2)
0220 *
0230     1 W-SRCTX     (A90)
0240     1 REDEFINE W-SRCTX /* <-- PRIMERA REDEFINICION
0250     2 W-NUM-REG   (B2)
0260     2 W-TEXTO    (A78)

Proxima pantalla listar desde 280_   Buscar: _____
-----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
SALIR LOCAL MAPA

```

Figura 3

## 4.2 PROGRAMAS:

Con independencia del concepto de programa visto al principio del apartado anterior, se entiende por *objeto tipo programa* un proceso constituido por un conjunto de instrucciones que se pueden ejecutar independientemente de otros objetos.

En este sentido, un objeto tipo programa, una vez compilado, pasa a denominarse **comando**, y en consecuencia puede invocarse exactamente igual que cualquier otro comando del sistema.

Bajo la plataforma Natural, los ejecutables pueden considerarse como **COMANDOS** de un programa ejecutable principal llamado NATURAL.

La instrucción Natural que provoca que se ejecute un programa es la sentencia **FETCH** o **FETCH RETURN**, aunque también puede ejecutarse cargando en el stack, o pila de comandos pendientes de ejecución, el nombre del programa a ejecutar mediante la sentencia:

**STACK [TOP] COMMAND** nombre\_pgm.

Cuando Natural detiene la ejecución de instrucciones, bien porque se haya ejecutado la sentencia **STOP**, bien porque haya terminado de ejecutar las sentencias de un programa, lo primero que hace NATURAL es examinar el **stack** para saber si tiene que ejecutar algún comando, y, si es así, carga el nuevo ejecutable y comienza la ejecución de sus sentencias.

Por ultimo, en este punto cabe resaltar que aunque se pueda ejecutar un programa vía **FETCH RETURN**, se aconseja usar los tipos de módulos subprogramas y subrutinas, y reservar los módulos tipo programa para los procesos de tipo programa principal.

### 4.3 SUBROUTINAS

Estas pueden ser **internas** y **externas**. Las subrutinas internas se encuentran definidas en el mismo modulo que el programa llamante, por lo que no requieren el paso de datos. Se corresponden con un conjunto de instrucciones agrupadas bajo un nombre, formando una rutina interna, de modo que pueda ejecutarse de 0 a n veces. Comparte todas las variables con éste, y su nombre puede tener hasta 32 caracteres, lo cual añade significado. (Ejemplo: Perform PASAR-FECHA-AAAAMMDD-A-DDMMAAAA).

En el caso de subrutinas externas, estas se encuentran definidas en un modulo distinto al del proceso llamante. Este modulo normalmente tiene por nombre el de la subrutina, aunque hay que tener presente que un modulo tipo subroutine puede contener varias subrutinas externas, por lo que se debe diferenciar entre nombre del modulo y nombre de la subrutina, y recordar que se invoca una subrutina externa por su nombre y no por el nombre del modulo que la contiene.

### 4.4 COPYCODE

De este tipo de objeto solo existe el fuente, ya que como indica su nombre, es un fragmento de código fuente que será incluido en el objeto que le referencia en el momento de la compilación.

Por sus características, este objeto se utiliza principalmente para activar teclas de función, controlar códigos de retorno o tratamientos de errores. Debe estar en la misma librería que el objeto que contiene el **INCLUDE**.

De este tipo, cabe comentar que las áreas de datos podrían considerarse como copycodes, pues al fin y al cabo solo contienen definiciones de variables. Esto no es totalmente cierto, ya que las áreas de datos permiten definir variables haciendo uso del editor especializado en este tipo de objetos, al tiempo que por el hecho de poder compilar un área de datos, se permite efectuar una serie de validaciones previas al proceso de compilación del programa, para así incorporarse a este posteriormente con el consiguiente ahorro de tiempo<sup>8</sup>. En cambio, los objetos tipo COPYCODE no permiten realizar una compilación previa ya que, entre otras cosas, faltan las definiciones de formato de las variables.

### 4.5 MAPA

Este objeto es invocado mediante la sentencia **INPUT**, y al ejecutarse presenta una pantalla que servirá de *interface* con el usuario para la captura y presentación de la información.

Este tipo, además, puede incluir llamadas a otros dos tipos: **HELPMAPAS** Y **HELPROUTINES**, al tiempo que permite incluir reglas de proceso.

Para su creación es recomendable el uso del **LAYOUT**, ya que esto proporciona comodidad y eficiencia. Este es un campo que figura en los parámetros de creación del mapa y permite incluir en el mapa que se esta inicializando, las definiciones del mapa referenciado en el campo LAYOUT (**figura 4**), lo cual permite homogeneizar entre otras cosas, el formato general o diseño de las pantallas de una instalación o aplicación, tales como cabeceras, PFs, etc, sin necesidad de estar dibujando cada una de estas.

De este tipo de objeto cabe comentar que la propia arquitectura de NATURAL libera la memoria ocupada por el modulo llamado cuando se regresa al programa llamante por terminarse la ejecución de aquel. Por tanto, podría darse el caso de que al ejecutarse en el programa principal un **REINPUT** como consecuencia de una validación no superada, el mapa al que se regresa, no se encuentre ya en el Buffer POOL, con lo que tendría que leerle de nuevo desde el fichero del sistema NATURAL.

---

<sup>8</sup>Se recuerda en este sentido que los programas NATURAL, no son ejecutables en el sentido amplio de la palabra, sino que solo pueden ejecutarse bajo la plataforma Natural, por lo que estos ejecutables, podrían considerarse como **COMANDOS** de un programa EJECUTABLE principal llamado NATURAL.

```

15:10:19          Define Map Settings for MAP          96-04-18

Delimiters          Format          Context
-----
Cls Att CD Del Page Size ..... 23          Device Check ....
T D BLANK Line Size ..... 100        WRITE Statement
T I ? Column Shift ... 0 (0/1)      INPUT Statement X
^ D _ Layout .....
^ I ) dynamic ..... N (Y/N)
^ N ^ Zero Print ..... N (Y/N)
M D & Case Default ... UC (UC/LC)
M I : Manual Skip ... N (Y/N)
O D + Decimal Char ... ,
O I ( Standard Keys .. Y (Y/N)
Justification .. L (L/R)
Print Mode .....
Control Var .....

Automatic Rule Rank 1
Profile Name .... SYSPROF

Filler Characters
-----
Optional, Partial ....
Required, Partial ....
Optional, Complete ...
Required, Complete ...

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
Help Exit Let

```

Figura 4

Por otra parte, como ya se ha dicho, este objeto puede incluir también **reglas de proceso** que se ejecutaran secuencialmente<sup>9</sup> por niveles una vez que se pulse *INTRO*, lo cual puede ser una gran ventaja o una gran desventaja, dependiendo del uso que se haga de las mismas.

- **Ventaja:** Puede usarse para eliminar validaciones, y en consecuencia código, del programa principal, tales como formatos de fechas, campos obligatorios, etc.

- **Desventaja:** Pues, en el caso de que se incluyan procesos de elaboración de datos en estas reglas, elimina la claridad de un procedimiento secuencial a la hora de buscar errores

#### 4.6 HELPROUTINE

Este tipo de objeto es de proceso, pues es una rutina que se invoca como información de ayuda desde un campo de un mapa. Debe haberse codificado utilizando el editor de programas, y se activa al escribir el carácter “?” en el campo.

La asignación de la ayuda, se realiza cumplimentando el atributo **HE**, tal y como muestra la **figura 5**.

```

^rr W-COD-ERROR                                     Fmt N4
-----
AD= OILT_____ ZP= OFF SG= OFF HE= 'LISOB-JOH'_____ Rls 0
NL= _____ CD= __ CV= _____ Mod Data
PM= _____ DY= _____
EM= _____

001 --010---+---+---+---030---+---+---+---050---+---+---+---070---+---
(XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXXXXXX
(XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXXXXXX
-----
SELEC REFERENCIA DIV IMPORTE TIPO FECHA ERR.
-----
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (E999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
>: X (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXX (XXXXXX (9999
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
HELP Mset Exit <--- ---> -- - + < > Let

```

Figura 5

<sup>9</sup>Para listar la secuencia de instrucciones solo hay que ejecutar el comando **LIST M nombre\_map**

La **figura 2** muestra el código de una HELPROUTINE, demostrando que es como una rutina normal, con la salvedad de que solo tiene en el área de parámetros como primer campo, el campo donde se encuentra definida la llamada.

#### **4.7 HELPTEXT o HELPMAPA**

Es un mapa que es invocado como información de ayuda, y por tanto se ha codificado utilizando el editor de mapas.

Se invoca exactamente igual que la HELPROUTINE, haciendo uso del atributo HE.

Se diferencia de la HELPROUTINE en que el HELPTEXTO es solo un mapa con información sobre el campo, pero no recibe ni pasa parámetros, mientras que la HELPROUTINE es una rutina que normalmente lista datos para poder elegir un elemento.

#### **4.8 DESCRIPCIÓN**

Este objeto recoge la descripción de un programa tal y como se encuentra definido en el diccionario de datos **PREDICT**. Solo puede utilizarse este tipo si el sistema tiene instalado PREDICT.

## 5. Próximo artículo

Una vez comentados los distintos objetos que facilitan la modularización de NATURAL, razones de espacio aconsejan dejar para el próximo mes la opinión que tiene el autor sobre cual debe ser el uso que se haga de los mismos.

## 6. Utilidad

La utilidad que acompaña este artículo, y cuyos fuentes se encuentran en el disco que acompaña a la revista, ha sido seleccionada por facilitar el seguimiento interactivo de los fuentes, permitiendo listar cualquier fuente, buscar cadenas, cambiar de fuente, etc. estando abierto a todo tipo de mejoras, tales como imprimir un fuente o crear un informe con los distintos objetos referenciados en él.

No es en absoluto compleja, ya que se limita a leer directamente el fuente desde el fichero del sistema, en lugar de invocar cualquier **comando del sistema** (programa proporcionado por el constructor) para que realice este cometido. Esto permite leer cualquier fuente de cualquier librería, saltándose incluso el control de **NATURAL SECURITY**, ya que se hace uso del concepto utilizado por NATURAL para almacenar sus objetos. Este consiste en considerar que el nombre del continente tiene una longitud de 18 bytes, de los que los 8 primeros especifican la librería, y los 8 siguientes especifican el nombre del fuente, tal y como muestra la **figura 3**.

La **figura 6** muestra el **diagrama de Flujo Modular** de la utilidad. Este diagrama, que se explicara con detalle en el próximo artículo, permite determinar de forma rápida cada uno de los módulos que intervienen en la aplicación cuando se hace uso de la regla **UN PROGRAMA UN MAPA**.

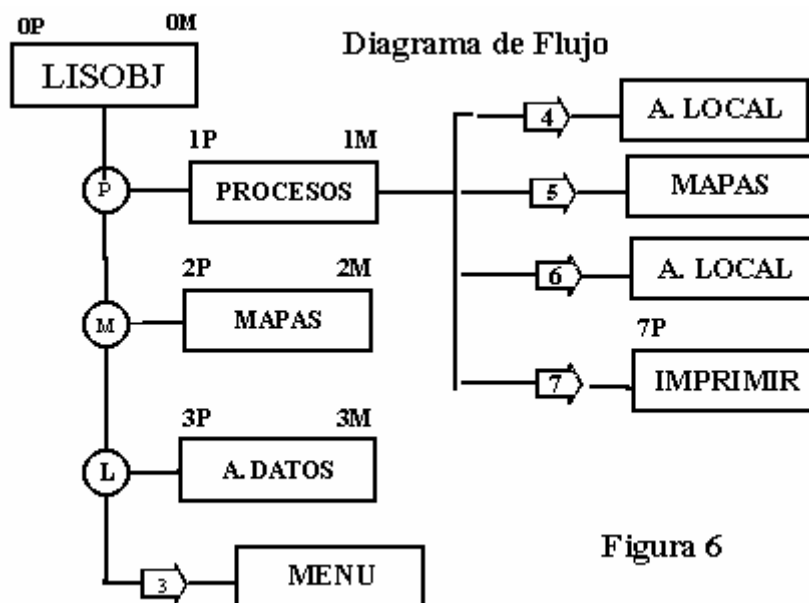


Figura 6