

Curso programación ABAP IV:

**Temario y
material de soporte**

TEMARIO DEL CURSO

1. INTRODUCCION

- 1.1 QUE ES UN ERP
- 1.2 MODULOS DE SAP
- 1.3 CONCEPTO DE MANDANTE
- 1.4 USUARIO Y PASS
- 1.5 MENU SAP, MENU USUARIO, FAVORITOS, TRANSACCIONES,
- 1.6 INTRODUCCIÓN AL SISTEMA BASE
- 1.7 R/3 SAP PORTALS BW, ETC

2. INTRODUCCION A LA PROGRAMACIÓN

- 2.1 DEFINICIÓN DE DATOS (Tipos de datos, variables, constantes, estructuras, rangos, field-symbols)
- 2.2 MODULARIZACIÓN (Subrutinas y includes)
- 2.3 VARIABLES DEL SISTEMA (SYST)
- 2.4 INSTRUCCIONES BÁSICAS
 - 2.4.1 **Asignaciones** (“=”, MOVE, MOVE-CORRESPONDING, CLEAR, OFFSET)
 - 2.4.2 **Operaciones condicionales** (EQ, NE, GE,, IF, CASE, Op. Condicionales Cadenas CS, CA...)
 - 2.4.3 **Iteración (bucles)** (DO, WHILE, CONTINUE, CHECK, EXIT)
 - 2.4.4 **Aritméticas** (SQRT, ADD, SUBTRACT, MULTIPLY, DIV, MOD)
 - 2.4.5 **Tratamiento de cadenas** (CONCATENATE, SHIFT, CONDENSE, TRANSLATE, REPLACE, STRLEN)
 - 2.4.6 **Formateo de valores** (WRITE TO, PACK, UNPACK)
 - 2.4.7 **Mensajes** (MESSAGE)
 - 2.4.8 **Comentarios de programas**
- 2.5 TABLAS INTERNAS
 - 2.5.1 **Declaración**
 - 2.5.2 **Creación de registros**
 - 2.5.3 **Lectura de registros**
 - 2.5.4 **Modificación de registros**
 - 2.5.5 **Borrado de registros**
 - 2.5.6 **Tratamiento de registros**
- 2.6 TABLAS DICCIONARIO
 - 2.6.1 **Declaración**
 - 2.6.2 **Creación de registros**
 - 2.6.3 **Selección de datos (SELECT ver todas las opciones)**
 - 2.6.4 **Lectura de registros por clave**
 - 2.6.5 **Modificación de registros**
 - 2.6.6 **Borrado de registros**
 - 2.6.7 **Tratamiento de registros**
- 2.7 DEBUGGING

3. DICCIONARIO

- 3.1 INTRODUCCIÓN AL DICCIONARIO
- 3.2 DOMINIO
- 3.3 ELEMENTO DE DATOS
- 3.4 TIPOS DE TABLAS (**transparentes, estructuras, vistas, cluster...**)
- 3.5 TABLAS (SE11)
 - 3.5.1 Creación de tablas**
 - 3.5.2 Opciones técnicas**
 - 3.5.3 Estructuras append**
 - 3.5.4 Índices**
 - 3.5.5 Utilidades base de datos**
 - 3.5.6 Generación actualización de tablas**
 - 3.5.7 Visualizar/Modificar contenido tablas (SE16, SE16N y SM30)**
 - 3.5.8 Claves Externas**
- 3.6 ESTRUCTURAS
- 3.7 VISTAS
- 3.8 MATCHCODE (**Ayudas de búsqueda**)
- 3.9 OBJETOS DE BLOQUEO

4. REPORTS

- 4.1 INTRODUCCIÓN (PLANOS E INTERACTIVOS)
- 4.2 EDITOR ABAP IV (SE38, SE80)
 - 4.2.1 **Documentación y estructuración de un listado plano.**
 - 4.2.2 **Definición de atributos de salida.**
 - 4.2.3 **Definición de tablas externas.**
 - 4.2.4 **Pantalla de selección.**
 - 4.2.5 **Elementos de texto**
 - *Símbolos de texto*
 - *Textos de selección*
 - *Títulos y cabeceras*
 - 4.2.6 **Sentencias de salida de datos**
 - 4.2.7 **Message**
 - 4.2.8 **Eventos**
 - *INITIALIZATION*
 - *START-OF-SELECTION*
 - *END-OF-SELECTION*
 - *TOP-OF-PAGE*
 - *END-OF-PAGE*
 - *AT SELECTION-SCREEN*
 - 4.2.9 **Eventos de rupturas de secuencia en tablas internas**
 - *AT FIRST ... ENDAT*
 - *AT LAST ... ENDAT*
 - *AT NEW campo ... ENDAT*
 - *AT END OF campo ... ENDAT*

- *ON CHANGE OF campo ... ENDON*

4.2.10 Bases de datos lógicas

4.2.11 FIELD-GROUPS

4.2.12 Programación de listados interactivos

- *Introducción*
- *Eventos (AT LINE-SELECTION, AT USER-COMMAND, CASE SY-UCOMM, AT PFn)*
- Sentencias de lectura y escritura de líneas (HIDE, READ LINE)
- Niveles de listados anidados
- Menu Painter
- Screen Painter

4.3 ALV GRID

5. PROGRAMACIÓN DE DIÁLOGO

5.1 INTRODUCCIÓN

5.2 MODULE POOLS

5.2.1 Introducción

5.2.2 Programa marco o principal

5.2.3 Atributos de la pantalla

5.2.4 Diseño gráfico

5.2.5 Status de pantalla (barra menús, teclas función, barra herramientas, barra pulsadores, ...)

5.2.6 Lógica del proceso

- *PBO*
- *PAI*
- *PROCESS ON HELP-REQUEST*
- *PROCESS ON VALUE-REQUEST*

5.2.7 Sentencias

- *SET PF-STATUS*
- *SET TITLEBAR*
- *MODULE*
- *LEAVE PROGRAM*
- *AT EXIT-COMMAND*
- *AT CURSOR-SELECTION*
- *MESSAGE*
- *FIELD*
- *CHAIN...ENDCHAIN*
- *ON INPUT*
- *ON CHAIN-INPUT*
- *ON REQUEST*
- *ON CHAIN-REQUEST*
- *ETC*

5.2.8 Asignación de transacciones (SE93)

5.2.9 Modificación dinámica de una pantalla (LOOP AT SCREEN)

5.2.10 TABLE CONTROL Y TABS

5.2.11 Secuencia de proceso de pantallas (SET SCREEN, LEAVE SCREEN, LEAVE TO SCREEN, CALL SCREEN, LEAVE PROGRAM)

5.2.12 Procesamiento de listados en pantallas

6-FUNCIONES

6.1 INTRODUCCIÓN

6.2 CREACIÓN DE UN GRUPO DE FUNCIONES

6.3 CREACIÓN DE UNA FUNCIÓN

6.4 PARÁMETROS DE UNA FUNCIÓN

6.4.1 Import

6.4.2 Export

6.4.3 Changing

6.4.4 Tablas

6.5 EXCEPCIONES

6.6 DATOS GLOBALES

6.7 CÓDIGO FUENTE

6.8 EJECUCIÓN

6.9 RFCs y BAPIs

7-LLAMADAS A PROGRAMAS Y GESTIÓN DE MEMORIA

7.1 INTRODUCCIÓN

7.2 PARÁMETROS DE MEMORIA SAP (SET, GET, DATOS PROPIOS)

7.3 SENTENCIAS DE LLAMADAS A PROGRAMAS (SUBMIT, CALL TRANSACTION)

7.4 INTERCAMBIO DE DATOS A TRAVÉS DE LA MEMORIA ABAP/4 (EXPORT, IMPORT, FREE MEMORY)

8-INTERFASES

8.1 INTRODUCCIÓN

8.2 TRATAMIENTO DE FICHEROS

8.3 INSTRUCCIONES

8.3.1 Lectura/Escritura de ficheros físicos (GUI_DOWNLOAD, GUI_UPLOAD)

8.3.2. Lectura ficheros servidor

- *OPEN DATASET*
- *READ DATASET*
- *TRANSFER*
- *CLOSE DATASET*
- *DELETE DATASET*

8.3.3 EXPLORAR FICHEROS (AL11)

8.3.4 TRANSACCIÓN FILE (FICHEROS LÓGICOS)

8.3.5 PROGRAMAS DE CARGA

8.3.6 BATCH INPUT

- ***Introducción***

- *Grabadora (SHDB).*
 - *BDC_OPEN_GROUP*
 - *BDC_INSERT*
 - *CLOSE_GROUP*
 - *Creación de juegos de datos*
 - *Tratamiento/Ejecución de juegos de datos*
 - *CALL TRANSACTION (OPCIONES)*
 - *DIRECT INPUT*
- 8.3.7 LEGACY SYSTEM MIGRATION WORKBENCH (LSMW)

9-FORMULARIOS

- 9.1 INTRODUCCIÓN
- 9.2 ESTRUCTURA DE UN FORMULARIO (PROG.IMPRESIÓN, FORMULARIO)
- 9.3 EDITOR DEL FORMULARIO (SE71)
 - 9.3.1 **Cabecera**
 - 9.3.2 **Páginas**
 - 9.3.3 **Ventanas**
 - 9.3.4 **Ventana página**
 - 9.3.5 **Formatos de párrafo**
 - 9.3.6 **Formatos caracteres**
 - 9.3.7 **Elementos de texto**
- 9.4 SAPSCRIPT
 - 9.4.1 **Cajas, líneas y sombreados**
 - 9.4.2 **Comandos de control**
 - 9.4.3 **Símbolos SAPScript**
 - 9.4.4 **Símbolos del sistema**
 - 9.4.5 **Campos generales de SAPScript**
 - 9.4.6 **Opciones de formato de los símbolos**
 - 9.4.7 **Formularios en varios idiomas**
 - 9.4.8 **Inclusión de gráficos**
- 9.5 PROGRAMA DE IMPRESIÓN
 - 9.5.1 **OPEN_FORM**
 - 9.5.2 **WRITE_FORM**
 - 9.5.3 **CLOSE_FORM**
 - 9.5.4 **ETC**
- 9.6 SMARTFORMS
 - 9.6.1 **Parametrización global**
 - 9.6.2 **Páginas y ventanas (MAIN)**
 - 9.6.3 **Elementos para control de flujo**

10-PROGRAMACIÓN ORIENTADA A OBJETOS

- 10.1 INTRODUCCIÓN
- 10.2 CLASES Y OBJETOS
- 10.3 METODOS Y PROPIEDADES
- 10.4 HERENCIA

- 10.5 POLIMORFISMO
- 10.6 TABLAS INTERNAS DE OBJETOS
- 10.7 EJEMPLOS

11- AMPLIACIONES SAP

- 11.1 CMOD y SMOD
- 11.2 USER EXIT
- 11.3 BADIS

12- RESTO DE INSTRUCCIONES ABAP

- 12.1 REPASO DE LAS INSTRUCCIONES ABAP

13- TABLAS ESTANDAR SAP

- 13.1 PRINCIPALES TABLAS ESTANDAR SAP

14- TRANSACCIONES ESTANDAR SAP

- 14.1 OBTENCIÓN DE LA AYUDA TÉCNICA (F1)
- 14.2 TRACE DEL SISTEMA (ST05)
- 14.3 TRANSPORTE (SE10 y STMS)
- 14.4 ANÁLISIS DE ERRORES (ST22)
- 14.5 SPOOL (SP01)
- 14.6 PROGRAMACIÓN DE JOBS (SM37)
- 14.7 SXDA
- 14.8 EDITOR SPLIT SCREEN (SE39)
- 14.9 OBJECT NAVIGATOR (SE80)
- 14.10 MENÚ ÁMBITO (SE43N)
- 14.11 QUERYS (SQ01, SQ2, SQ03)
- 14.12 WORKFLOW
- 14.13 VERIFICACIÓN AMPLIADA

1. Introducción

1.1 Qué es un ERP

Un ERP (Enterprise Resource Planning) permite la integración de los procesos de negocio y sistemas para alcanzar una amplia eficiencia operacional de la empresa. Además mejora el funcionamiento de un negocio consolidando las operaciones de una empresa en una única base de datos, una sola aplicación y una única interfaz de usuario.

1.2 Módulos de SAP

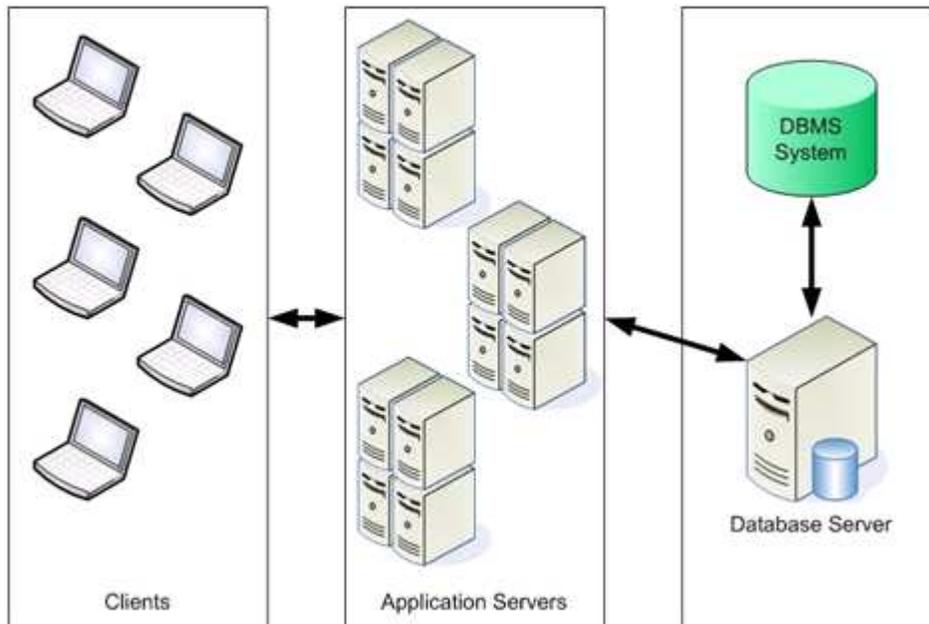
SAP (Systems Application Products in Data Processing) R/3 es un Enterprise Resource Planning

Sus ventajas se resumen en las siguientes:

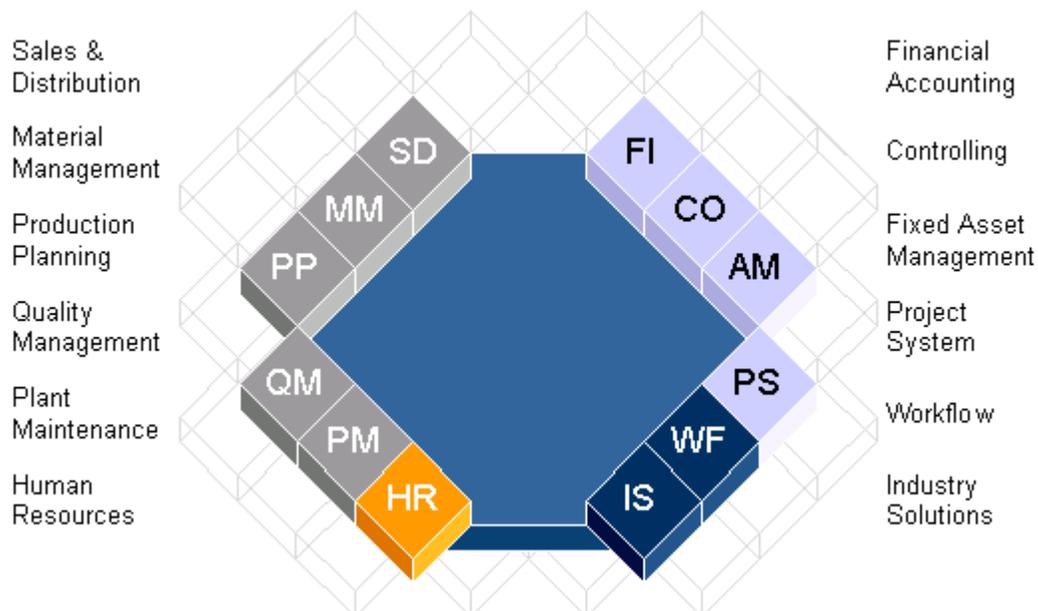
- Integración
- Flexibilidad
- Procesamiento de datos en tiempo real
- Diseñado para todo tipo de negocio

Presenta una arquitectura de tres capas: Cliente, Servidor de aplicación y Servidor de Base de Datos

3 Tier Client/Server Architecture



Se presentan en SAP diferentes módulos. Los vemos con un esquema:



- **Sales and Distribution (SD).** Ventas y distribución.
Da soporte a las tareas y actividades llevadas a cabo en ventas, entregas y facturaciones.
- **Materials Management (MM)** Gestión de materiales.
Da soporte a la consecución y funciones de inventario, como la compra, la gerencia de inventario.
- **Production Planning (PP)**
Se utiliza para planear y controlar las actividades de la fabricación de una empresa. Este módulo incluye; las listas de material, encaminamientos, centros de trabajo, ventas y la planificación de operaciones, la planificación de exigencias material, el control de planta, órdenes de producción, coste de producto, etc.
- **Quality Management (QM)** Es un control de calidad y proporciona el control de la fabricación y la consecución.
- **Plant Maintenance (PM)** Es un proceso de fabricación de complejo donde el equipo puede ser reconstruido y el mantenimiento de servicio proporcionado. El Mantenimiento de Planta esta muy relacionado con PP.
- **Human Resources (HR).** Recursos Humanos es un sistema completo integrado para apoyar la planificación y el control de actividades de personal.
- **Financial Accounting (FI).** Finanzas. Diseñado para dirección automatizada y generación de informes externos del libro de contabilidad general, cuentas por cobrar, cuentas por pagar y otras cuentas del sublibro de contabilidad.

- **Controlling (CO)** Representa el flujo de coste de la empresa

1.3 Concepto de mandante

El concepto de mandante se refiere a diferentes escenarios de datos independientes unos de otros con los que el sistema trabaja. Es simplemente un área de trabajo dentro de un ambiente SAP (producción, integración y desarrollo).

1.4 Usuario y Password

Usuarios Sistema Ayuda

SAP

Clave acceso nueva

Mandante 205

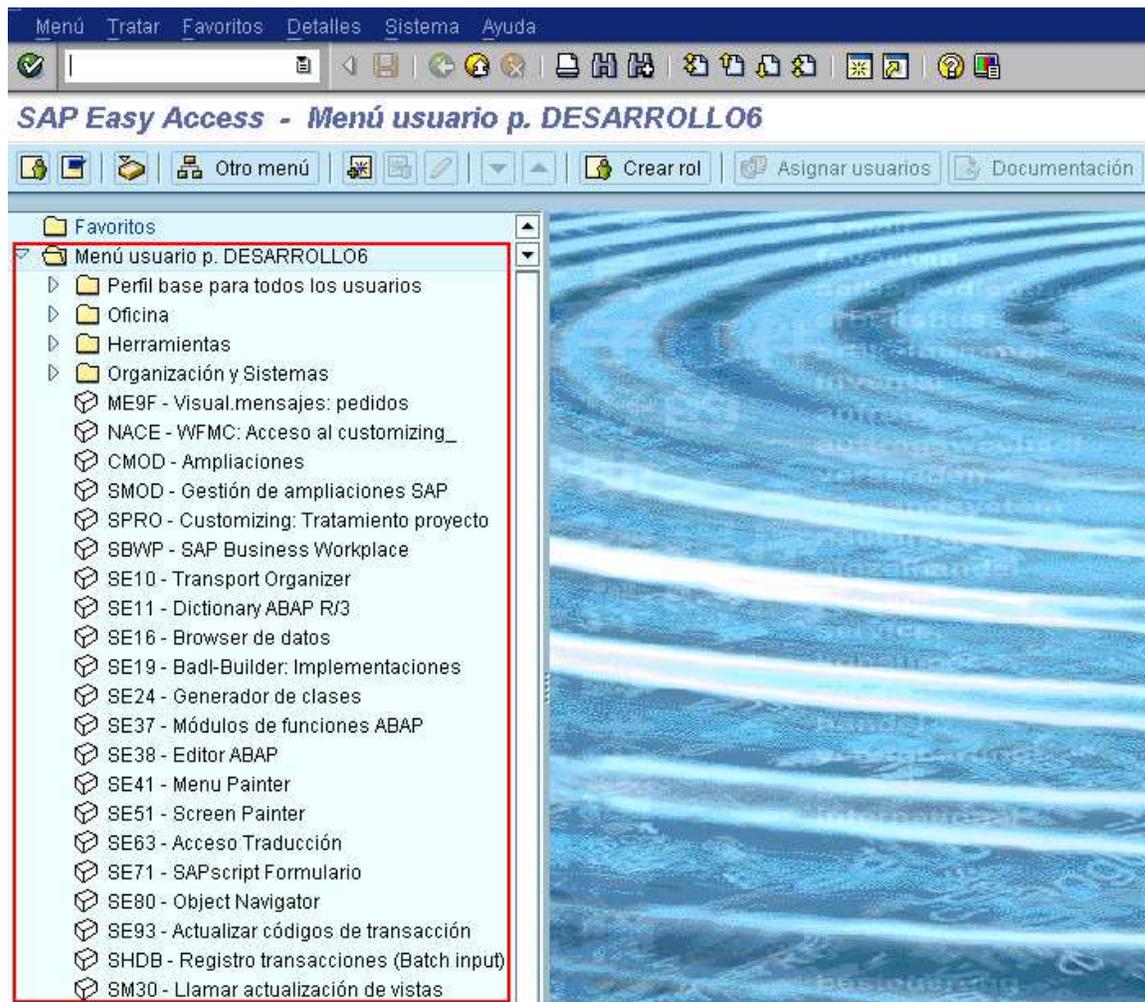
Usuarios desarro11o6

Clave de acceso *****

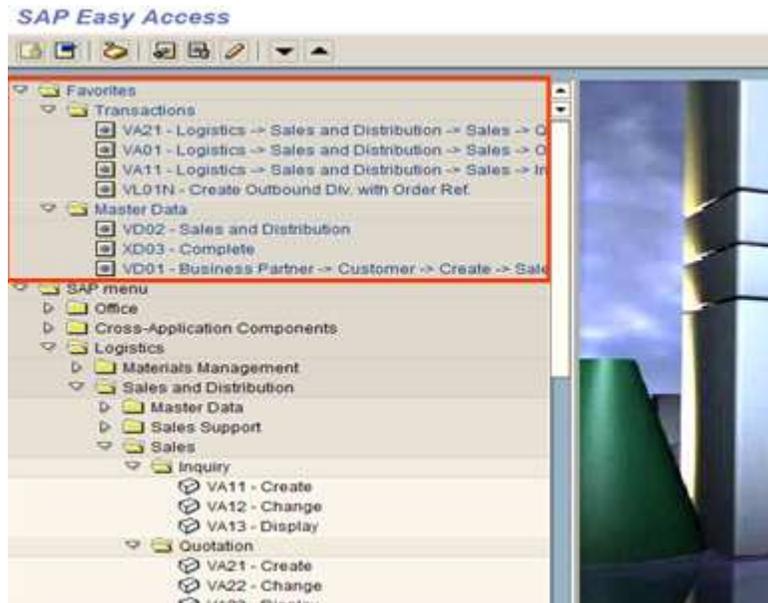
Idioma

1.5 Menu SAP, Menú Usuario, Favoritos transacciones

Menú de usuario



Favoritos, transacciones.



2. Introducción a la programación

2.1 Definición de datos

2.1.1 Tipos de datos.

Los tipos de datos que se pueden utilizar en ABAP /4 son los siguientes:

Tipos	Long. por Defecto	Longitud posible	Valor inicial	Descripción	
C	1	1-320	ESPACIO	Texto	
F	8	8	S	Punto flotante	
I	4	4	0.0E+00	Entero	
N	1	1-320	'0000'	Texto numérico	
P	8	1-16	0	Número Empaquetado	
X	1	1-298	x'00'	Hexadecimal	
D	8	8	70	00000000	Fecha YYYYMMDD
T	6	6	000000	Hora HHMMSS	

2.1.2 Variables

Las variables se definen con la sentencia **DATA**.

DATA <variable>(longitud) TYPE <tipo> VALUE <valor>.

Si no se indica lo contrario las variables se definirán por defecto de tipo carácter (Texto) con longitud 1.

DATA D_CHARACTER. “Se creará una variable de texto con longitud 1

DATA D_CHARACTER (8). “Se creará una variable de texto con longitud 8

Con la cláusula TYPE podemos especificar el tipo de dato de las variables.

DATA: D_NUMERO_CAR (5) TYPE N,

D_NUMERO TYPE I,

D_FECHA TYPE D.

Utilizando el carácter “:” detrás de una instrucción se encadenan varias ejecuciones de la misma instrucción separadas por el carácter “,”.

Con el parámetro **VALUE** podemos inicializar la variable con un valor distinto al que tiene por defecto.

DATA D_CONTADOR TYPE I VALUE 1.

Con la cláusula **LIKE** podemos declarar una variable con los mismos atributos de longitud y tipo que un campo del diccionario de datos o que otra variable definida previamente.

DATA D_ACREEDOR LIKE LFA1-LIFNR.

2.1.3 Constantes

Las constantes se definen con la sentencia **CONSTANTS**.

CONSTANTS <constante>(longitud) TYPE <tipo> VALUE <valor>.

Para la definición de constantes se aplican las mismas reglas que para la definición de variables pero siempre hay que darle un valor inicial que no podrá ser modificado durante la ejecución del programa.

CONSTANTS C_ACREEDOR LIKE LFA1-LIFNR VALUE ‘JOSE’.

2.1.4 Estructuras

Una estructura es un conjunto de campos relacionados lógicamente y se define con la sentencia **DATA**:

DATA: BEGIN OF <estructura>,

...

END OF <estructura>.

```
DATA: BEGIN OF R_PROVEEDOR,  
      CODIGO LIKE LFA1-LIFNR,  
      NOMBRE LIKE LFA1-NAME1,  
      CIUDAD(20) LIKE LFA1-ORT01,  
      FECHA TYPE D,  
      END OF R_PROVEEDOR.
```

También se puede incluir en la definición de una estructura la de una tabla del diccionario de datos o una estructura definida previamente en el programa utilizando la instrucción **INCLUDE STRUCTURE**.

```
DATA BEGIN OF R_SOCIEDADES OCCURS 10.  
      INCLUDE STRUCTURE T001.  
DATA: CONTADOR(20) TYPE N,  
      END OF R_SOCIEDADES.
```

Estas instrucciones crean una estructura con todos los campos de la tabla del diccionario T001 y el campo CONTADOR como último campo. Para hacer referencia posteriormente a los campos de la estructura se especificará el nombre de la estructura y del campo separados por un guión.

R_PROVEEDOR-NOMBRE = 'JOSE'.

2.1.5 Rangos

Los rangos son tablas internas con una estructura determinada que se utilizan para almacenar rangos de valores para un tipo de campo y se definen con la sentencia **RANGES**:

RANGES: <rango> FOR <campo>.

La estructura de un rango se compone de los siguientes campos:

SIGN: Es un campo de tipo carácter de una posición que indica si los valores especificados en el rango se incluyen (valor "I") o se excluyen (valor "E") del mismo.

OPTION: Es un campo de tipo carácter de dos posiciones que indica el operador lógico que une los valores especificados en el rango.

LOW: Valor inicial del rango. Este campo tiene el mismo tipo que el campo al que hace referencia el rango.

HIGH: Valor final del rango. Este campo tiene el mismo tipo que el campo al que hace referencia el rango.

Los rangos se utilizan para chequear valores válidos en sentencias condicionales del campo al que hace referencia. Si un rango no contiene ningún registro, asume que cualquier valor del campo al que hace referencia es válido.

```
TABLES: LFA1.  
RANGES G_PROVEEDOR FOR LFA1-LIFNR.
```

```
G_PROVEEDOR-SIGN = "1".
G_PROVEEDOR-OPTION = "BT". "Operador ENTRE
G_PROVEEDOR-LOW = "1".
G_PROVEEDOR-HIGH = "9".
APPEND G_PROVEEDOR.
```

El rango creado hace referencia a los valores del campo LFA1-LIFNR comprendidos entre el valor "1" y "9" ambos inclusive.

2.1.6 FIELD-SYMBOLS

Los FIELD-SYMBOLS son campos simbólicos que se pueden referenciar a un campo concreto en tiempo de ejecución del programa, y se definen con la sentencia **FIELD-SYMBOLS**:

FIELD-SYMBOLS <nombre>. "(los caracteres "<" y ">" se deben especificar).

Utilizando la sentencia ASSIGN se asigna un campo al FIELD-SYMBOL y, a partir de ese momento, todas las operaciones que se realicen sobre el FIELD-SYMBOL harán referencia al campo asignado.

```
FIELD-SYMBOLS: <CAMPO>.
DATA: D_CONTADOR_1 TYPE I,
      D_CONTADOR_2 TYPE I.
```

```
ASSIGN D_CONTADOR_1 TO <CAMPO>.
<CAMPO> = <CAMPO> + 1.
ASSIGN D_CONTADOR_2 TO <CAMPO>.
<CAMPO> = <CAMPO> + 2.
```

Después de ejecutar las sentencias anteriores la variable D_CONTADOR_1 contendrá el valor "1" y la variable D_CONTADOR_2 contendrá el valor "2".

2.2 Modularización

2.2.1 Subrutinas.

Las subrutinas son conjuntos de instrucciones que pueden ser llamadas desde el programa en diversas ocasiones, evitando así su codificación cada vez que se necesiten ejecutar y dando además mayor claridad al programa.

Para definir una subrutina se utilizan las sentencias **FORM** y **ENDFORM**, que se especifican respectivamente al inicio y final de las instrucciones que forman la subrutina.

Para realizar una llamada a la misma se utiliza la sentencia **PERFORM**.

PERFORM EJEMPLO. " Llamada a la subrutina

...

FORM EJEMPLO. " Definición de subrutina

```
...  
INSTRUCCIONES ABAP.  
...  
ENDFORM.
```

El programa principal y la subrutina se podrán comunicar mediante parámetros que pueden ser pasados por valor o por referencia, siendo recomendable especificar el tipo de dato al definirlos.

- El paso de parámetros **por referencia** se realiza de la siguiente manera:

```
...  
PERFORM <nombre> USING parametro1 parametro2 ...  
...  
FORM <nombre> USING parametro1 parametro2 ...  
...  
ENDFORM.
```

Los parámetros que son pasados a la subrutina en la llamada (ACTUALES) actualizan el valor de los parámetros definidos en la subrutina (FORMALES).

```
DATA: D_NUMERO_1 TYPE I VALUE 1,  
      D_NUMERO_2 TYPE I VALUE 2.  
  
PERFORM SUMAR USING D_NUMERO_1 D_NUMERO_2.  
  
FORM SUMAR USING PS_NUMERO TYPE I  
      PS_CANTIDAD TYPE I.  
      PS_NUMERO = PS_NUMERO + PS_CANTIDAD.  
ENDFORM.
```

Al finalizar la ejecución de la subrutina las variables tendrán estos valores:

```
D_NUMERO_1 => 3  
D_NUMERO_2 => 2
```

- Pase de parámetros **por valor**.

```
...  
PERFORM <nombre> USING parametro1 parametro2 ...  
...  
FORM <nombre> USING VALUE(parametro1) VALUE(parametro2) ...  
...  
ENDFORM.
```

Los parámetros que son pasados a la subrutina en la llamada no se actualizan con el valor de los parámetros definidos en la subrutina.

```
DATA: D_NUMERO_1 TYPE I VALUE 1,  
      D_NUMERO_2 TYPE I VALUE 2.
```

```
PERFORM SUMAR USING D_NUMERO_1 D_NUMERO_2.
```

```
FORM SUMAR USING PS_NUMERO TYPE I  
      VALUE(PE_CANTIDAD) TYPE I.  
  PE_CANTIDAD = 3.  
  PS_NUMERO = PS_NUMERO + PE_CANTIDAD.  
ENDFORM.
```

Al finalizar la ejecución de la subrutina las variables tendrán estos valores:
D_NUMERO_1 => 4
D_NUMERO_2 => 2

Utilizando la cláusula **STRUCTURE** se puede pasar como parámetro una estructura y hacer referencia a sus campos en la subrutina.

```
DATA: BEGIN OF R_PROV,  
      CODIGO LIKE LFA1-LIFNR,  
      NOMBRE LIKE LFA1-NAME1,  
      END OF R_PROV.
```

```
PERFORM INFORMAR_PROV USING R_PROV.
```

```
FORM INFORMAR_PROV USING PS_PROV STRUCTURE R_PROV.  
  PS_PROV-NOMBRE = 'JOSE'.  
ENDFORM.
```

También se puede pasar como parámetro una tabla interna por referencia con la cláusula **TABLES**.

```
DATA: D_CONTADOR TYPE I VALUE 1.  
DATA: BEGIN OF I_PROV OCCURS 0,  
      CODIGO LIKE LFA1-LIFNR,  
      NOMBRE LIKE LFA1-NAME1,  
      END OF I_PROV.
```

```
PERFORM INFORMAR_PROV TABLES I_PROV  
      USING D_CONTADOR.
```

```
FORM INFORMAR_PROV TABLES PS_I_PROV STRUCTURE I_PROV  
      USING VALUE(PE_CONTADOR).  
  PS_I_PROV-CODIGO = PE_CONTADOR.  
  PS_I_PROV-NOMBRE = 'JOSE'.  
  APPEND PS_I_PROV. " Inserta el registro en la tabla interna  
ENDFORM.
```

Al finalizar la ejecución de la subrutina se habrá insertado el registro en la tabla interna I_PROV.

2.2.2 Includes

Los INCLUDES son unos programas que contienen definiciones que pueden ser utilizadas en otros programas insertando en estos la sentencia INCLUDE <nombre>.

Este tipo de programas no son ejecutables directamente, han de estar incluidos en otros para que puedan ser ejecutado el código que contienen.

2.3 Variables del sistema

2.3.1 Tabla SYST

ABAP/4 tiene algunas variables internas que se van actualizando automáticamente y que pueden ser utilizadas en los programas. Estas variables se encuentran almacenadas en la tabla del diccionario SYST y para acceder a ellas se les debe añadir el prefijo "SY-".

2.3.2 Variable SY-SUBRC

La variable más importante es SY-SUBRC, que devuelve el resultado de la ejecución de las instrucciones ABAP/4, siendo 0 el código de retorno de una ejecución correcta y otros valores en caso de error (estos valores de error dependen de la instrucción ABAP que se este ejecutando).

Variables del sistema más utilizadas.

CAMPO	CONTENIDO
ABCDE	Alfabeto (A, B, C, D,)
COLNO	Columna actual en la creación de una lista
CPAGE	Nº de la página actual
CPPROG	Nombre del programa principal
CUCOL	Posición del cursor, columna
CURROW	Posición del cursor, línea
DATUM	Fecha del sistema
DBNAME	Base de datos lógica usada en Report
DYNGR	Grupo de dynpros al que pertenece la dynpro actual
DYNNR	Nº de la pantalla actual
FDPOS	Lugar de hallazgo de un string
FMKEY	Menú de teclas de función
INDEX	Cantidad de ejecuciones del LOOP
LILLI	Nº de línea en la lista actual
LINCT	Cantidad de líneas de la lista
LINNO	Línea actual en la creación de una lista
LINSZ	Longitud de la línea de una lista

LISEL	Línea seleccionada (Listado interactivo)
LISTI	Nº de la línea de la lista actual
LISTI	Índice de selección de las listas
LOOPC	Cantidad de líneas LOOP en Step-Loop de una Dynpro
LSIND	Índice de las listas secundarias
LSTAT	Información del status por nivel de lista
MSGID	Identificador del mensaje
MSGNO	Nº del mensaje
MSGTY	Tipo del mensaje (E, I, W, ...)
MSGV1 – 4	Variables en mensaje
PAGCT	Límite de columnas de la lista en la sentencia REPORT
PAGNO	Página actual en creación de lista
PFKEY	Status actual de teclas de función
PRBIG	Impresión: portada de selección
PRIMM	Impresión: salida inmediata
PRREL	Impresión: borrar tras salida
REPID	Nombre del programa Abap / 4
SCOLS	Total de columnas en la pantalla
SROWS	Total de líneas en la pantalla
STACO	Margen del listado, columna
STARO	Margen del listado, línea
STEP	Índice del Step-Loop
STEPL	Nº de la línea Loop en Step de una Dynpro
SUBRC	Nº del error al ejecutar una operación
TABIX	Línea actual de tabla interna
TCODE	Código de la transacción actual
TFILL	Nº actual de entradas en la tabla interna
TITLE	Título del programa Abap / 4
TLENG	Tamaño de la línea de una tabla interna
TMAXL	Cantidad máxima de entradas en tabla interna
TNAME	Nombre de la tabla interna después de un acceso
TOCCU	Parámetros OCCURS en tabla interna
TTABC	Nº de la última línea de tabla interna leída
TVAR0 – 9	Variable de texto para elementos de texto de Abap / 4
UCOMM	Entrada de función en campos comando (Pulsadores, ...)
ULINE	Línea horizontal
UNAME	Nombre del usuario
UZEIT	Hora del sistema
VLINE	Línea vertical
WILLI	Nº de línea en la ventana actual
WINCO	Posición del cursor en la ventana, columna
WINDI	Índice de la línea de la ventana actual
WINRO	Posición del cursor en la ventana, línea
WINSL	Línea de la ventana seleccionada
WTITL	Indicador para cabecera estándar de página

2.4 Instrucciones básicas.

2.4.1 Asignación

Existen diversas formas de asignar valores a una variable en ABAP/4:

- **Asignación directa**
Se asigna un valor a una variable directamente utilizando el operador “=”.
<variable> = valor.

El valor se puede especificar como un literal, el contenido de una constante o el contenido de una variable.

- **Sentencia MOVE.**
Asigna un valor a una variable.
MOVE <valor> TO <variable>.
- **Sentencia MOVE-CORRESPONDING.**
Mueve el contenido de los campos de una estructura a los campos que tengan el mismo nombre de otra estructura.
MOVE-CORRESPONDING <estructura_origen> TO <estructura_destino>.
- **Sentencia CLEAR.**
Inicializa el contenido de un campo a su valor inicial según el tipo de dato (espacios en el tipo carácter, cero en el tipo entero, etc.).
CLEAR <campo>.
- **OFFSETS.**
Se puede hacer referencia parcial a una parte de un campo indicando el número de posiciones que hay hasta la posición en la que se comienza a referenciar utilizando el operador '+', y la cantidad de posiciones que se referencian entre paréntesis.
MOVE <campo1>+<posiciones>(longitud) TO <campo2>.

```
DATA: D_CHARACTER_1(8) TYPE C VALUE 'ABCDEFGH',  
      D_CHARACTER_2(4) TYPE C.
```

```
D_CHARACTER_2 = D_CHARACTER_1+2(4) .
```

Al ejecutar la sentencia de asignación la variable D_CHARACTER_2 tendrá el valor 'CDEF'.

2.4.2 Condicionales

Las sentencias condicionales permiten ejecutar una serie de instrucciones solamente cuando se cumplan una serie de condiciones.

- Operadores condicionales.
Las condiciones se construyen utilizando los siguientes operadores condicionales:
- <valor> 1 EQ <valor2>.
La condición se cumple si <valor1> es igual que <valor2>. (También se puede utilizar el carácter “=” en lugar de “EQ”).
- <valor1> NE <valor2>.
La condición se cumple si <valor1> es distinto que <valor2>. (También se pueden utilizar los caracteres “<” en lugar de “NE”).
- <valor1> GT <valor2>.
La condición se cumple si <valor1> es mayor que <valor2>. También se puede utilizar el carácter “>” en lugar de “GT”).
- <valor1> LT <valor2>.
La condición se cumple si <valor1> es menor que <valor2>. También se puede utilizar el carácter “<” en lugar de “LT”).
- <valor1> GE <valor2>.
La condición se cumple si <valor1> es mayor o igual que <valor2>. También se pueden utilizar los caracteres “>=” en lugar de “GE”).
- <valor1> LE <valor2>.
La condición se cumple si <valor1> es menor o igual que <valor2>. También se pueden utilizar los caracteres “<=” en lugar de “LE”).
- <valor1> BETWEEN <valor2> AND <valor3>.
La condición se cumple si <valor1> está comprendido entre los valores <valor2> y <valor3>.
- <valor1> IS INITIAL.
La condición se cumple si <valor1> tiene valor inicial según el tipo de dato (espacios en el tipo carácter, cero en el tipo entero, etc.).
- <valor> IN <rango>.
La condición se cumple si <valor> tiene un valor comprendido en el rango <rango>.

- Operadores lógicos.
Los operadores lógicos permiten combinar condiciones:
- <condición1> AND <condición2> .
La condición formada se cumple si las dos condiciones <condición1> y <condición2> se cumplen.
- <condición1> OR <condición2> .
La condición formada se cumple si al menos una de las dos condiciones se cumple.
- NOT <condición>.
La condición formada se cumple si la condición <condición> es falsa.

- Sentencias **IF** <condición> ... **ENDIF**.
Las instrucciones delimitadas por las sentencias IF y ENDIF solamente se ejecutan si la condición especificada se cumple.

```
FORM ESCRIBIR USING VALUE(PE_TEXTO)
      VALUE(PE_ESCRIBIR).
```

```
IF NOT PE_ESCRIBIR IS INITIAL.
*   Se escribe el texto en la pantalla
    WRITE PE_TEXTO.
ENDIF.
```

```
ENDFORM.
```

Esta subrutina escribirá por pantalla el texto recibido en el parámetro PE_TEXTO si se recibe un valor distinto de espacios en el parámetro PE_ESCRIBIR (la sentencia de escritura en pantalla WRITE se verá en el manual de reporting).

- Sentencias **CASE** <variable> ... **ENDCASE**.

Permiten ejecutar diferentes grupos de sentencias en función del valor contenido en <variable>, delimitando las sentencias con la cláusula WHEN <valor>.

El valor "OTHERS" comprende todos los valores que no hayan sido especificados en las anteriores cláusulas WHEN.

```
FORM ESCRIBIR USING VALUE(PE_TEXTO)
      VALUE(PE_ALINEACION).
```

```
CONSTANTS: C_L_CENTRADO(1) TYPE C VALUE 'C',
            C_L_IZQUIERDA(1) TYPE C VALUE 'I',
            C_L_DERECHA(1) TYPE C VALUE 'D'.
```

```
CASE PE_ALINEACION.
  WHEN C_L_CENTRADO.
*     Se escribe el texto en la pantalla centrado
    WRITE PE_TEXTO CENTERED.
  WHEN C_L_IZQUIERDA.
*     Se escribe el texto en la pantalla justificado a
*     la izquierda
    WRITE PE_TEXTO LEFT-JUSTIFIED.
  WHEN C_L_DERECHA.
*     Se escribe el texto en la pantalla justificado a
*     derecha
    WRITE PE_TEXTO RIGHT-JUSTIFIED.
  WHEN OTHERS.
*     Se escribe el texto en la pantalla sin justificar
    WRITE PE_TEXTO.
ENDCASE.
ENDFORM.
```

Esta subrutina escribirá por pantalla el texto recibido en el parámetro PE_TEXTO según el valor recibido en el parámetro PE_ALINEACION:

Si se recibe el valor 'C' se escribe centrado.

Si se recibe el valor 'I' se escribe justificado a la izquierda.

Si se recibe el valor 'D' se escribe justificado a la derecha.

Si se recibe cualquier otro valor se escribe sin justificar.

- Operadores condicionales para cadenas.
Utilizando los siguientes operadores se pueden construir condiciones específicas para cadenas de texto:
- <cadena1> CO <cadena2>.
La condición se cumple si todos los caracteres contenidos en <cadena1>, incluyendo espacios en blanco, existen en <cadena2>.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá la longitud de <cadena1>, sino contendrá el OFFSET del primer carácter de <cadena1> que no existe en <cadena2>.
- <cadena1> CN <cadena2>.
La condición se cumple si alguno de los caracteres contenidos en <cadena1>, incluyendo espacios en blanco, no existe en <cadena2>.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá el OFFSET del primer carácter de <cadena1> que no existe en <cadena2>, sino contendrá la longitud de <cadena1>.
- <cadena1> CA <cadena2>.
La condición se cumple si alguno de los caracteres contenidos en <cadena1>, incluyendo espacios en blanco, existe en <cadena2>.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá el OFFSET del primer carácter de <cadena1> que existe en <cadena2>, sino contendrá la longitud de <cadena1>.
- <cadena1> NA <cadena2>.
La condición se cumple si ninguno de los caracteres contenidos en <cadena1>, incluyendo espacios en blanco, existe en <cadena2>.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá la longitud de <cadena1>, sino contendrá el OFFSET del primer carácter de <cadena1> que existe en <cadena2>.
- <cadena1> CS <cadena2>.
La condición se cumple si <cadena1> contiene <cadena2>.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá el OFFSET del primer carácter de <cadena2> que existe en <cadena1>, sino contendrá la longitud de <cadena1>.

- `<cadena1> NS <cadena2>`.
La condición se cumple si `<cadena1>` no contiene `<cadena2>`.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá la longitud de `<cadena1>`, sino contendrá el OFFSET del primer carácter de `<cadena2>` que existe en `<cadena1>`.
- `<cadena> CP <patrón>`.
La condición se cumple si `<cadena>` contiene el patrón de búsqueda `<patrón>`. Este patrón permite utilizar el carácter "*" para representar cualquier cadena de caracteres y el carácter "+" para representar cualquier carácter.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá el OFFSET del primer carácter de `<cadena2>` que existe en `<cadena1>`, sino contendrá la longitud de `<cadena1>`.

Ejemplos: La condición 'Jose Luis Díaz' CP 'Jose*Diaz' es verdadera y devuelve el valor '0' en la variable SY-FDPOS.
La condición 'JoseLuisDíaz' CP 'Jose+Luis' es falsa y devuelve el valor '12' en la variable SY-FDPOS.

- `<cadena> NP <patrón>`.
La condición se cumple si `<cadena>` no contiene el patrón de búsqueda `<patrón>`.
Si la condición es cierta, la variable del sistema SY-FDPOS contendrá la longitud de `<cadena1>`, sino contendrá el OFFSET del primer carácter de `<cadena2>` que existe en `<cadena1>`.

2.4.3 Iteración (bucles)

- Sentencias **DO ... ENDDO**.
Ejecuta las sentencias delimitadas por DO y ENDDO hasta que se ejecute una sentencia de salida como EXIT.

```
DATA: D_CONTADOR TYPE I.
```

```
DO.  
  D_CONTADOR = D_CONTADOR + 1.  
  IF D_CONTADOR = 10.  
    EXIT.  
  ENDIF.  
ENDDO.
```

Las sentencias del bucle DO se ejecutarán hasta que se ejecuta la sentencia EXIT, que finaliza la ejecución del bucle después de haber ejecutado las sentencias 10 veces. La variable D_CONTADOR tendrá el valor 10. La variable del sistema SY-INDEX contiene el número de vueltas del bucle. Con la cláusula TIMES se puede especificar el número máximo de vueltas del bucle.

DATA: D_CONTADOR TYPE I.

```
DO 3 TIMES.  
  D_CONTADOR = D_CONTADOR + 1.  
  IF D_CONTADOR = 10.  
    EXIT.  
  ENDIF.  
ENDDO.
```

Las sentencias del bucle DO se ejecutarán 3 veces y la variable D_CONTADOR tendrá el valor 3.

- Sentencias **WHILE** <condición> ... **ENDWHILE**.

Ejecuta las sentencias delimitadas por WHILE y ENDWHILE hasta que no se cumpla la condición especificada o se ejecute una sentencia de salida como EXIT.

DATA: D_CONTADOR TYPE I.

```
WHILE D_CONTADOR < 10.  
  D_CONTADOR = D_CONTADOR + 1.  
ENDWHILE.
```

Las sentencias del bucle WHILE se ejecutarán hasta que se deja de cumplir la condición cuando la variable D_CONTADOR alcanza el valor 10. La variable del sistema SY-INDEX contiene el número de vueltas del bucle.

- Sentencia **CONTINUE**.

Esta sentencia se utiliza dentro de un bucle y provoca que se finalice la vuelta actual sin ejecutar las sentencias posteriores y se inicie la ejecución de la siguiente vuelta del bucle.

DATA: D_CONTADOR TYPE I.,
 D_NUMERO TYPE I.

```
WHILE D_CONTADOR < 10.  
  D_CONTADOR = D_CONTADOR + 1.  
  IF D_CONTADOR > 4.  
    CONTINUE.  
  ENDIF.  
  D_NUMERO = D_CONTADOR.  
ENDWHILE.
```

Las sentencias del bucle WHILE se ejecutarán 10 veces pero solamente se ejecuta la asignación a la variable D_NUMERO solo se ejecuta las 4 primeras vueltas. Al finalizar

la ejecución del bucle la variable D_CONTADOR tendrá el valor 10 y la variable D_NUMERO el valor 4.

- Sentencia **CHECK** <condicion>.

Esta sentencia, al igual que la sentencia CONTINUE, provoca el final de la ejecución de la vuelta actual en un bucle sin ejecutar las sentencias posteriores e iniciando la ejecución de la siguiente vuelta cuando no se cumpla la condición especificada.

```
DATA: D_CONTADOR TYPE I.,  
      D_NUMERO TYPE I.
```

```
WHILE D_CONTADOR < 10.  
  D_CONTADOR = D_CONTADOR + 1.  
  CHECK D_CONTADOR < 5.  
  D_NUMERO = D_CONTADOR.  
ENDWHILE.
```

Las sentencias del bucle WHILE se ejecutarán 10 veces pero solamente se ejecuta la asignación a la variable D_NUMERO solo se ejecuta las 4 primeras vueltas. Al finalizar la ejecución del bucle la variable D_CONTADOR tendrá el valor 10 y la variable D_NUMERO el valor 4. La ejecución de esta sentencia fuera de un bucle finaliza inmediatamente la ejecución de la subrutina donde se ejecuta.

```
FORM SUMAR USING PE_NUMERO_1 TYPE I  
                PE_NUMERO_2 TYPE I  
                PS_SUMA TYPE I.  
  CHECK PE_NUMERO_1 > 0 AND  
        PE_NUMERO_2 > 0.  
  PS_SUMA = PE_NUMERO_1 + PE_NUMERO_2.  
ENDFORM.
```

Esta subrutina solamente sumará números positivos ya que si alguno de los números que recibe como parámetros es negativo se finaliza la ejecución de la subrutina sin ejecutar el resto de las sentencias.

- Sentencia EXIT.
Provoca la salida de un bucle o el final de la ejecución de la subrutina donde se ejecuta.

2.4.4 Aritméticas

- Sentencia SQRT.
Devuelve la raíz cuadrada de un valor.
<campo> = SQRT(<valor>).
- Sentencia ADD.

Suma un valor al contenido de un campo.
ADD <valor> TO <campo>.

También se puede utilizar el operador '+'.
<campo> = <campo> + <valor>.

- Sentencia SUBTRACT.

Resta un valor al contenido de un campo.
SUBTRACT <valor> FROM <campo>.

También se puede utilizar el operador '-'.
<campo> = <campo> - <valor>.

- Sentencia MULTIPLY.

Multiplica por un valor el contenido de un campo.
MULTIPLY <campo> BY <valor>.

También se puede utilizar el operador '*'.
<campo> = <campo> * <valor>.

- Sentencia DIV.

Devuelve el cociente de una división entera.
<campo> = <valor1> DIV <valor2>.

El operador '/' devuelve el cociente con decimales.
<campo> = <valor1> / <valor2>.

- Sentencia MOD.

Devuelve el resto de una división entera.
<campo> = <valor1> MOD <valor2>.

2.4.5 Tratamiento de cadenas

- Sentencia CONCATENATE.

Concatena cadenas en un campo.
CONCATENATE <cadena1> <cadena2> INTO <campo>.

Con la cláusula SEPARATED BY se puede especificar un separador entre las cadenas.

- Sentencia SHIFT.

Elimina caracteres por la izquierda de un campo.
SHIFT <campo>.

```
DATA: D_CADENA(8) VALUE 'ABCDEFGH'.  
SHIFT D_CADENA.
```

La variable D_CADENA tendrá el valor 'BCDEFGH'.

Con la cláusula CIRCULAR se provoca que los caracteres que se eliminan por la izquierda se inserten por la derecha de la cadena.

```
DATA: D_CADENA(8) VALUE 'ABCDEFGH'.  
SHIFT D_CADENA CIRCULAR.
```

La variable D_CADENA tendrá el valor 'BCDEFGHA'.

- Sentencia CONDENSE.

Elimina los espacios en blanco por la izquierda de una cadena y reduce los espacios intermedios consecutivos a un solo espacio.
CONDENSE <campo>.

```
DATA: D_CADENA(25) VALUE ' JOSE LUIS'.  
CONDENSE D_CADENA.
```

La variable D_CADENA tendrá el valor 'JOSE LUIS'.

Con la cláusula NO-GAPS se eliminan todos los espacios de la cadena.

- Sentencia TRANSLATE.

Permite convertir a mayúsculas o minúsculas una cadena con las cláusulas TO UPPER CASE y TO LOWER CASE.
TRANSLATE <campo> TO UPPER/LOWER CASE.

- Sentencia REPLACE.

Reemplaza una parte de una cadena por otra cadena.
REPLACE <subcadena> WITH <subcadena_nueva> INTO <campo>.

```
DATA: D_CADENA(10) VALUE 'JOSE LUIS'.  
REPLACE 'JOSE' WITH 'JUAN' INTO D_CADENA.
```

La variable D_CADENA tendrá el valor 'JUAN LUIS'.

- Sentencia STRLEN.

Devuelve la longitud de una cadena sin tener en cuenta los espacios finales por la derecha.

```
DATA: D_CADENA(25) VALUE 'JOSE LUIS  ',  
      D_LONGITUD TYPE I.  
D_LONGITUD = STRLEN( D_CADENA ).
```

La variable D_LONGITUD tendrá el valor 9.

2.4.6 Formateo de valores

- Sentencia WRITE TO.
Escribe un valor en un campo permitiendo formatearlo con diferentes opciones de formato.
WRITE <valor> TO <campo> <opciones de formato>.
- Sentencia PACK.
Almacena un valor en formato empaquetado en el campo especificado.
PACK <valor> TO campo.
- Sentencia UNPACK.
Desempaqueta un valor y lo almacena con ceros a la izquierda en el campo especificado.
UNPACK <valor> TO <campo>.

2.4.7 Mensajes

En los programas ABAP se pueden mostrar mensajes creados previamente en una clase de mensajes utilizando la instrucción **MESSAGE**.

```
MESSAGE <tipo>NNN(<clase>)
```

Existen los siguientes tipos de mensajes:

- **Informativos** (tipo "I"): Se visualiza el mensaje en una ventana y continua la ejecución del programa al pulsar la tecla INTRO.
- **Éxito** (Success, tipo "S"): Se visualiza el mensaje en la barra de status en la siguiente pantalla y continua con la ejecución del programa al pulsar la tecla INTRO.
- **Error** (Error, tipo "E"): Se visualiza el mensaje en la barra de status e interrumpe el programa en función del tipo de programa (listado, module pool, etc.) al pulsar la tecla INTRO.
- **Advertencia** (Warning, tipo "W"): Se visualiza el mensaje en la barra de status y, si se pulsa la tecla ESCAPE permite realizar modificaciones en función del tipo de programa, y si se pulsa la tecla INTRO continua con la ejecución del programa.
- **Finalización** (Abend, tipo "A"): Se visualiza el mensaje en la barra de status y finaliza la ejecución del programa al pulsar la tecla INTRO.
- **Salida** (eXit, tipo "X"): Finaliza la ejecución del programa provocando un error en tiempo de ejecución (DUMB).

Para tratar los mensajes de una clase de mensajes se ejecuta la opción de menú 'Herramientas → Workbench Abap/4 → Desarrollo → Entorno de programación → Mensajes'.

2.4.8 Comentarios

En el código de los programas se deberán introducir comentarios que ayuden a comprender su lógica de proceso y así facilitar su mantenimiento posterior. Existen dos tipos de comentarios:

- **Comentario en línea:** Al especificar el carácter “ en una línea de programa, todos los literales que se escriban a su derecha hasta el final de la línea se considerarán comentarios.
- **Línea de comentario:** Al especificar el carácter * en la primera posición de una línea de texto, todos los literales de la línea se considerarán comentarios.

2.5 Tratamiento de tablas internas

Las tablas internas se almacenan en memoria y no en el diccionario de datos. Es frecuente su utilización para almacenar datos seleccionados de las tablas de la base de datos para procesarlos posteriormente, permitiendo ordenar dichos datos de diversas formas y tenerlos disponibles en todo momento, evitando accesos a la base de datos que penalizarían el rendimiento de los programas.

2.5.1 Declaración

Las tablas internas se declaran con la sentencia **DATA**.

```
DATA: BEGIN OF <tabla> OCCURS n,  
...  
          END OF <tabla>.
```

La cláusula **OCCURS** determina el número de registros de la tabla interna que se almacenarán en memoria principal, no el tamaño de la tabla, almacenándose el resto de registros en un área de paginación. Hay que tener especial cuidado al establecer este número porque si se define demasiado pequeño provoca un acceso más lento a la tabla y si es muy grande puede provocar que el área de almacenamiento destinado por SAP (ROLL AREA) se agote. Si no estamos seguros del número a especificar es recomendable asignarle cero para indicar que lo gestione SAP.

También se puede utilizar la instrucción **INCLUDE STRUCTURE** en la declaración de tablas internas de la misma forma que en las estructuras.

```
DATA: BEGIN OF I_PROVEEDORES OCCURS 0,  
      CODIGO LIKE LFA1-LIFNR,  
      NOMBRE LIKE LFA1-NAME1,  
      CIUDAD LIKE LFA1-ORT01,  
      FECHA TYPE D,  
      END OF I_PROVEEDORES.
```

Al declarar una tabla interna se define automáticamente una línea de cabecera con la estructura de la tabla. Esta línea de cabecera es una estructura con el nombre de la tabla interna que nos permite recuperar, modificar o borrar registros de la misma.

2.5.2 Creación de registros

La sentencia APPEND permite crear una entrada al final de una tabla interna con los datos almacenados en la línea de cabecera de la tabla. La variable del sistema SY-TABIX almacenará el índice del nuevo registro creado.

```
FORM AÑADIR_MATERIAL USING VALUE(PE_CODIGO) LIKE I_MATERIALES-  
CODIGO
```

```
    VALUE(PE_NOMBRE) LIKE I_MATERIALES-NOMBRE  
    VALUE(PE_CANTIDAD) LIKE I_MATERIALES-CANTIDAD.
```

```
MOVE: PE_CODIGO TO I_MATERIALES-CODIGO,  
      PE_NOMBRE TO I_MATERIALES-NOMBRE,  
      PE_CANTIDAD TO I_MATERIALES-CANTIDAD.  
APPEND I_MATERIALES.
```

```
ENFORM.
```

Esta subrutina crea entradas en la tabla de materiales con los datos recibidos como parámetros.

La sentencia COLLECT permite crear entradas en una tabla interna acumulando el contenido de los campos numéricos (tipos P, I o F).

Si no existe ningún registro en la tabla interna en el que coincidan los datos no numéricos almacenados en la línea de cabecera, se añadirá un nuevo registro en la tabla, sino se acumulará el contenido de los campos numéricos almacenados en la línea de cabecera en el registro. La variable del sistema SY-TABIX almacenará el índice del nuevo registro creado o del registro acumulado.

```
PERFORM ACUMULAR_MATERIAL USING '1'  
    'MATERIAL'  
    10.
```

```
PERFORM ACUMULAR_MATERIAL USING '2'  
    'MATERIAL'  
    20.
```

```
PERFORM ACUMULAR_MATERIAL USING '1'  
    'MATERIAL'  
    5.
```

```
FORM ACUMULAR_MATERIAL USING VALUE(PE_CODIGO) LIKE I_MATERIALES-  
CODIGO
```

```
    VALUE(PE_NOMBRE) LIKE I_MATERIALES -NOMBRE  
    VALUE(PE_CANTIDAD) LIKE I_MATERIALES-CANTIDAD.
```

```
MOVE: PE_CODIGO TO I_MATERIALES-CODIGO,  
      PE_NOMBRE TO I_MATERIALES-NOMBRE,  
      PE_CANTIDAD TO I_MATERIALES-CANTIDAD.  
COLLECT I_MATERIALES.
```

```
ENDFORM.
```

Al finalizar la ejecución la tabla de materiales almacenará las siguientes entradas:

CODIGO	NOMBRE	CANTIDAD
1	MATERIAL	15
2	MATERIAL	20

2.5.3 Lectura de registros

La sentencia READ TABLE permite leer un registro de una tabla interna especificando un argumento de búsqueda con la cláusula WITH KEY <campo1> = <valor1> <campo2> = <valor2>...

Si se logra leer el registro se almacenará en la línea de cabecera de la tabla, la variable del sistema SY-SUBRC contendrá el valor '0' y la variable SY-TABIX el índice del registro leído.

Si no se logra leer el registro la línea de cabecera de la tabla no se modificará, la variable del sistema SY-SUBRC contendrá un valor distinto de '0' y la SY-TABIX el valor '0'.

```
READ TABLE I_MATERIALES WITH KEY CODIGO = '1'.
```

La línea de cabecera contendrá los siguientes valores (en función de los registros añadidos en el ejemplo anterior de creación de registros):

CODIGO	NOMBRE	CANTIDAD
1	MATERIAL	15

La variable SY-SUBRC almacenará el valor '0'.

La variable SY-TABIX almacenará el valor '1'.

2.5.4 Modificación de registros

La sentencia MODIFY permite modificar un registro de una tabla con los datos almacenados en la línea de cabecera especificando el índice del registro con la cláusula INDEX.

Si se logra modificar el registro la variable del sistema SY-SUBRC contendrá el valor '0'.

```
READ TABLE I_MATERIALES WITH KEY CODIGO = '1'.
```

```
IF SY-SUBRC = 0.
```

```
I_MATERIALES-CANTIDAD = 25.  
MODIFY I_MATERIALES INDEX SY-TABIX.  
ELSE.  
  WRITE 'NO EXISTE EL MATERIAL'.  
ENDIF.
```

El primer registro de la tabla se habrá modificado con los valores almacenados en la línea de cabecera (en función de los registros utilizados en el ejemplo anterior de creación de registros):

CODIGO	NOMBRE	CANTIDAD
1	MATERIAL	25

La variable SY-SUBRC almacenará el valor '0'.
La variable SY-TABIX almacenará el valor '1'.

Se pueden realizar modificaciones masivas en una tabla interna utilizando la cláusula TRANSPORTING <campo1> <campo2> ... WHERE <campo> = <valor>. El valor almacenado en la línea de cabecera de los campos especificados se modificará en todos los registros que cumplan la condición especificada.

```
CLEAR I_MATERIALES-CANTIDAD.  
MODIFY TABLE I_MATERIALES  
  TRANSPORTING CANTIDAD  
  WHERE CODIGO > '0'  
  AND CODIGO < '101'.
```

Se modificará con el valor '0' el campo cantidad en todos los materiales de la tabla cuyo código este comprendido entre '1' y '100'.

2.5.5 Borrado de registros

La sentencia DELETE permite borrar un registro de una tabla interna especificando el índice del registro con la cláusula INDEX.

Si se logra borrar el registro, la variable del sistema SY-SUBRC contendrá el valor '0'.

```
READ TABLE I_MATERIALES WITH KEY CODIGO = '1'.  
IF SY-SUBRC = 0.  
  DELETE I_MATERIALES INDEX SY-TABIX.  
ELSE.  
  WRITE 'NO EXISTE EL MATERIAL'.  
ENDIF.
```

El registro del material con código '1' se ha borrado de la tabla interna. La variable SY-SUBRC almacenará el valor '0'.

Especificando una condición de borrado con la cláusula WHERE se borrarán todas las entradas de la tabla interna que cumplan la condición (está cláusula es incompatible con la cláusula INDEX).

También se pueden borrar todas las entradas de una tabla interna con la instrucción **REFRESH** <tabla>

2.5.6 Tratamiento de registros

La sentencia LOOP ... ENDLOOP permite tratar individualmente los registros de la tabla interna, generando un bucle en el que se va informando cada registro en la línea de cabecera y ejecutando las instrucciones especificadas entre las dos sentencias en cada vuelta.

Utilizando la cláusula WHERE <condicion> se puede especificar una condición para tratar solamente una parte de los registros de la tabla.

```
FORM TOTALIZAR_MATERIALES USING VALUE(PE_CODIGO) LIKE
I_MATERIALES-CODIGO.
      PS_CANTIDAD LIKE I_MATERIALES-CANTIDAD.
CLEAR PS_CANTIDAD.
LOOP AT I_MATERIALES WHERE CODIGO = PE_CODIGO.
  PS_CANTIDAD = PS_CANTIDAD + I_MATERIALES-CANTIDAD
ENDLOOP.
ENDFORM.
```

La sentencia **SORT** ordena los registros de la tabla interna, se pueden especificar el /los campo(s) por los que hay que ordenar así como el orden ascendente o descendente.

```
SORT I_MATERIALES BY CODIGO DESCENDING.
```

Ordena los registros de la tabla interna por código de material de forma descendente. Por defecto, los registros de la tabla interna se ordenan de forma ascendente por la clave por defecto, compuesta por los campos no numéricos de la tabla.

La sentencia **DESCRIBE TABLE** nos permite obtener las características de la tabla interna, entre ellas el número de registros que contiene la tabla utilizando la cláusula **LINES** <variable> .

2.6 Tratamiento de tablas del diccionario de datos

El diccionario de datos está perfectamente integrado con el entorno de desarrollo y cualquier modificación que se realice en una tabla del diccionario se reflejará automáticamente en los programas que la utilicen.

2.6.1 Declaración

Las tablas del diccionario de datos se declaran con la sentencia **TABLES**.

TABLES: <tabla1>, <tabla2>.

Al declarar una tabla del diccionario se define automáticamente una línea de cabecera con la estructura de la tabla. Esta línea de cabecera es una estructura con el nombre de la tabla que nos permite recuperar, modificar o borrar registros en la misma.

2.6.2 Creación de registros

La sentencia INSERT permite crear una entrada con los datos almacenados en la línea de cabecera de la tabla.

No se podrá añadir un registro si ya existe uno en la tabla con la misma clave primaria o con la misma clave de un índice de valores únicos.

Si la inserción del registro es correcta la variable SY-SUBRC devolverá '0' y si se produce un error en la inserción porque ya existe un registro con la misma clave devolverá el valor '4'.

```
FORM AÑADIR_MATERIAL USING VALUE(PE_CODIGO) LIKE I_MATERIALES-  
CODIGO  
                        VALUE(PE_NOMBRE) LIKE I_MATERIALES-NOMBRE  
                        VALUE(PE_CANTIDAD) LIKE I_MATERIALES-CANTIDAD.
```

```
MOVE: PE_CODIGO TO MATERIALES-CODIGO,  
      PE_NOMBRE TO MATERIALES-NOMBRE,  
      PE_CANTIDAD TO MATERIALES-CANTIDAD.  
INSERT MATERIALES.  
IF SY-SUBRC <> 0.  
  WRITE 'ERROR'.  
ENDIF.
```

ENFORM.

Esta subrutina crea entradas en la tabla del diccionario MATERIALES con los datos recibidos como parámetros.

2.6.3 Lectura de registros por clave

La sentencia SELECT con la cláusula SINGLE permite leer un registro de una tabla del diccionario de datos especificando su clave primaria como condición de búsqueda.

```
SELECT SINGLE * FROM <TABLA>  
  WHERE <condiciones>.
```

Si se logra leer el registro, se almacenará en la línea de cabecera de la tabla y la variable del sistema SY-SUBRC contendrá el valor '0', sino la variable del sistema SY-

SUBRC contendrá un valor distinto de '0' y la línea de cabecera de la tabla no se modificará.

Si se especifica la cláusula INTO, el registro leído se almacenará en la estructura especificada en lugar de almacenarse en la línea de cabecera.

```
SELECT SINGLE * FROM <TABLA>
                INTO <registro>
WHERE <condiciones>.
```

Se pueden recuperar campos específicos del registro enumerándolos en lugar de especificar el carácter "*", siendo necesario en ese caso especificar la cláusula INTO con una estructura que coincida exactamente con los campos seleccionados o enumerando los campos de destino (entre paréntesis si son varios).

```
SELECT SINGLE <campo1> <campo2> FROM <TABLA>
                                INTO (<campo1>, <campo2>)
WHERE <condiciones>.
```

Si no se especifica la clave primaria completa de la tabla como condición de búsqueda, se leerá el primer registro que cumpla la condición.

```
SELECT SINGLE *
FROM MATERIALES
WHERE CODIGO = 1.
```

Se seleccionará en la cabecera de la tabla el registro del material que contenga el valor "1" en el campo CODIGO, que es el campo que forma la clave primaria de la tabla.

Se puede seleccionar un registro de una tabla y bloquearlo para que no pueda ser modificado por otros procesos utilizando la cláusula SINGLE FOR UPDATE y especificando obligatoriamente la clave completa de la tabla como criterio de selección.

2.6.4 Modificación de registros

La sentencia UPDATE permite modificar un registro de una tabla del diccionario con los datos almacenados en la línea de cabecera, no debiendo estar modificados los valores de los campos clave de la tabla.

```
UPDATE <tabla>.
```

Utilizando la cláusula SET se podrán especificar los campos que se van a modificar, pudiendo modificar también los campos clave de la tabla y no teniendo en cuenta los valores almacenados en la línea de cabecera. Al utilizar esta cláusula se deberá tener especial cuidado, ya que actualizará todos los registros de la tabla a menos que se especifique la cláusula WHERE para delimitar los registros que queremos modificar.

```
UPDATE <tabla> SET <campo1> = <valor1> <campo2> = <valor2>
                WHERE <condiciones>.
```

Después de ejecutar una sentencia UPDATE, la variable del sistema SY-SUBRC contendrá el valor "0" si se ha logrado modificar al menos un registro y la variable SY-DBCNT contendrá el número de registros que han sido modificados.

```
UPDATE MATERIALES
SET CANTIDAD = CANTIDAD + 1
WHERE CODIGO > 0
AND CODIGO < 101.
```

Esta sentencia incrementa en una unidad el campo CANTIDAD en todos los materiales que tengan un valor comprendido entre 1 y 100 en el campo CODIGO.

2.6.5 Borrado de registros

La sentencia DELETE permite borrar un registro de una tabla cuya clave primaria coincida con la almacenada en la línea de cabecera. Si se logra borrar el registro, la variable del sistema SY-SUBRC contendrá el valor '0'.

```
MATERIALES-CODIGO = '1'.
DELETE MATERIALES.
```

Borrará el registro correspondiente al material cuyo código es '1'.

También se pueden borrar uno o más registros de una tabla del diccionario especificando una condición de borrado.

```
DELETE <tabla> WHERE <condiciones>.
```

2.6.6 Tratamiento de registros

Las sentencias SELECT...ENDSELECT permiten tratar individualmente los registros de una tabla del diccionario, generando un bucle en el que se va informando cada registro en la línea de cabecera y ejecutando las instrucciones especificadas entre las dos sentencias en cada vuelta.

```
FORM TOTALIZAR_MATERIALES
  USING VALUE(PE_CODINI) LIKE Z_MATERIALES-CODIGO
  VALUE(PE_CODFIN) LIKE Z_MATERIALES-CODIGO
  PS_TOTAL LIKE Z_MATERIALES-CANTIDAD.
```

```
SELECT * FROM Z_MATERIALES
  WHERE CODIGO >= PE_CODINI
  AND CODIGO <= PE_CODFIN.
```

```
  PS_TOTAL = PS_TOTAL + Z_MATERIALES-CANTIDAD.
```

```
ENDSELECT.
ENDFORM.
```

Esta subrutina totalizará en el parámetro de salida PS_TOTAL la cantidad de los materiales almacenados en la tabla del diccionario Z_MATERIALES, cuyo código esté comprendido entre los recibidos en los parámetros de entrada PE_CODINI y PE_CODFIN.

Cuando se van a tratar grandes cantidades de registros de una tabla del diccionario, es preferible almacenar los registros en una tabla interna con la cláusula INTO TABLE <tabla_interna> y realizar un tratamiento posterior de dicha tabla. Esta cláusula provoca la realización de un sólo acceso de lectura a la tabla del diccionario, en lugar de realizar una lectura por cada registro recuperado, optimizándose así el tiempo de ejecución del programa. En este caso no se genera un bucle para el tratamiento de los registros, por lo que no se utiliza la sentencia ENDSELECT.

La tabla interna especificada en la cláusula INTO TABLE debe tener la misma estructura que los campos recuperados de la tabla del diccionario, sino es así, se puede utilizar en su lugar la cláusula INTO CORRESPONDING FIELDS OF TABLE <tabla_interna> para almacenar los valores recuperados de la tabla del diccionario en los campos de la tabla interna que tengan el mismo nombre y tipo.

En un bucle SELECT...ENDSELECT no se puede ejecutar ninguna instrucción que provoque una actualización de la base de datos (CALL SCREEN, CALL TRANSACTION, MESSAGE, COMMIT-WORK, etc), ya que se produciría un error en tiempo de ejecución del programa. Al procesar los datos utilizando una tabla interna se elimina esa limitación.

- DEBUGGING (**FALTA**)

3. Diccionario

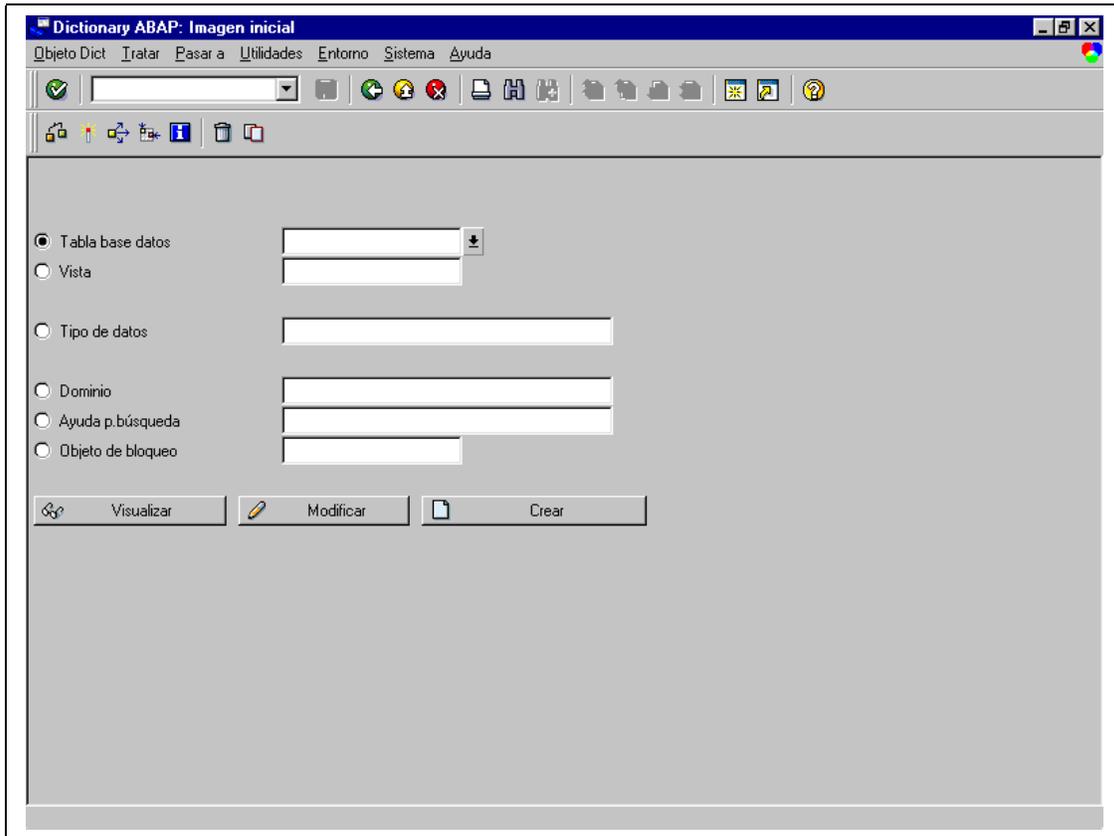
El Diccionario de datos lo componen todas las definiciones de datos, tipos de datos así como las tablas de la base de datos en la que se almacenan los datos.

El diccionario, lo componen todos los objetos tanto los Estándares como los desarrollados a medida.

Para llegar al menú de gestión del diccionario:

Ruta de acceso: (En el menú principal de SAP) Herramientas → Workbench ABAP4 → Desarrollo → Dictionary (SE11).

Llegaremos a la pantalla principal del diccionario:



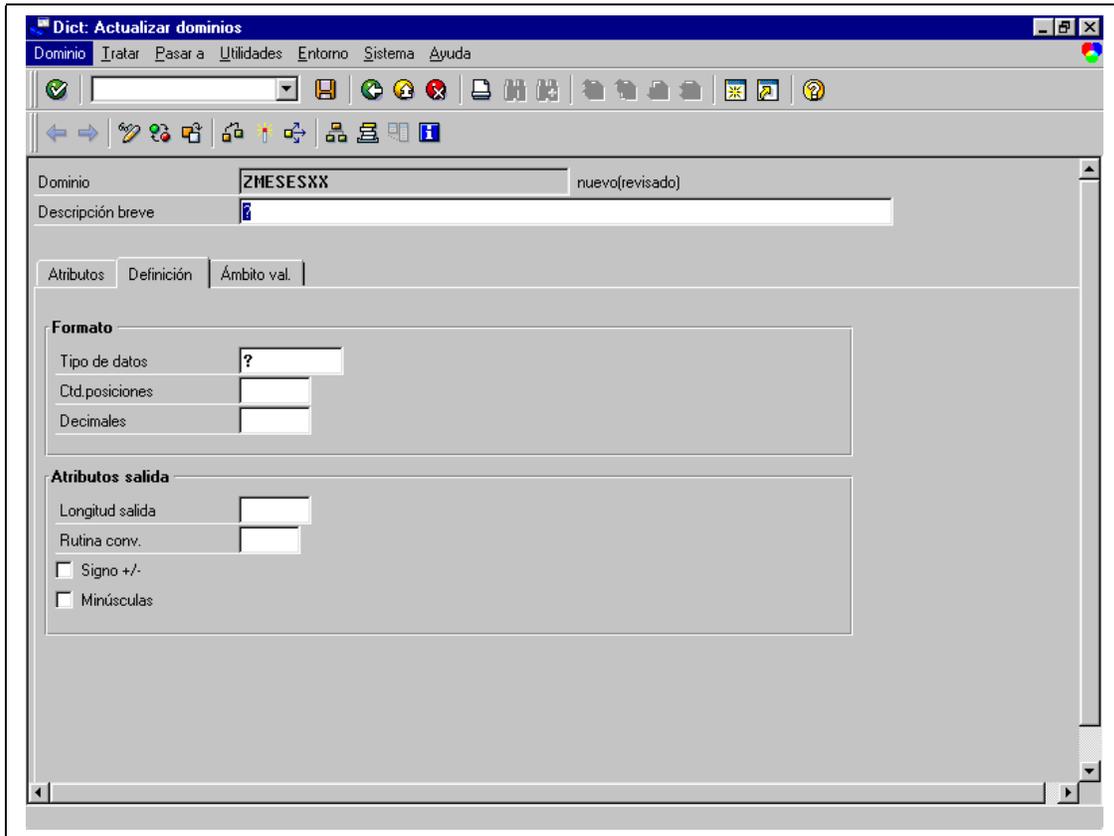
3.2 Dominios.

Un dominio es el objeto que define las características técnicas de un atributo. Mediante el dominio se definen el tipo de datos, longitud, valores posibles, propiedades de salida (Por pantalla, impresión...)...

Creación / Modificación / Visualización.

En el menú diccionario, introducimos el nombre del Dominio de datos que vamos a crear, *en nuestro caso ZMESESXX*, seleccionamos la opción "Dominios". Pulsamos el botón de Crear  (El procedimiento es similar para visualizar  y modificar ), también podemos acceder a estas opciones a través del menú 'Objeto Dict '

Llegaremos a la siguiente pantalla:

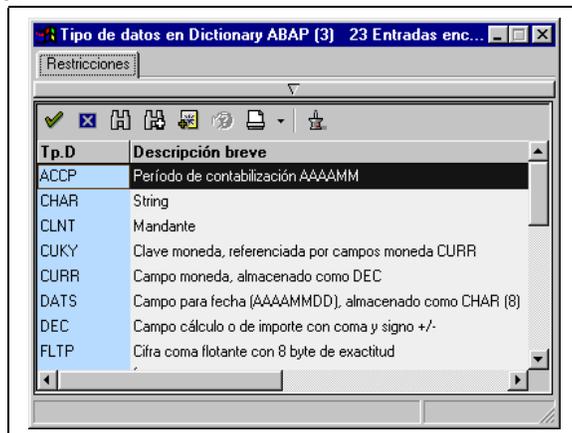


En la pestaña *Definición*:

Descripción breve: Descripción del dominio a crear, ha de ser un texto explicativo del dominio. *En nuestro caso escribiremos 'Meses de Año'.*

Tipo de datos: Seleccionaremos uno de los tipos existentes eligiendo el que mejor se adapte a las características del objeto que estamos definiendo. *En nuestro caso seleccionaremos 'CHAR'*

Los tipos de datos existentes se pueden visualizar (F4) nos mostrará la siguiente ventana:



Longitud: Se corresponderá al tamaño deseado. *En nuestro caso pondremos 10.*

Los elementos del marco *Atributos de salida*, varían dependiendo del tipo de datos seleccionado así por ejemplo para el tipo CHAR tendremos la opción de minúsculas y para un tipo CURR tendremos la opción del signo...

Longitud de salida: Representa la longitud en la que se va a representar el valor a la hora de imprimirse en un informe, visualizarse en una pantalla, etc. *(Dejamos en valores propuestos 10)*

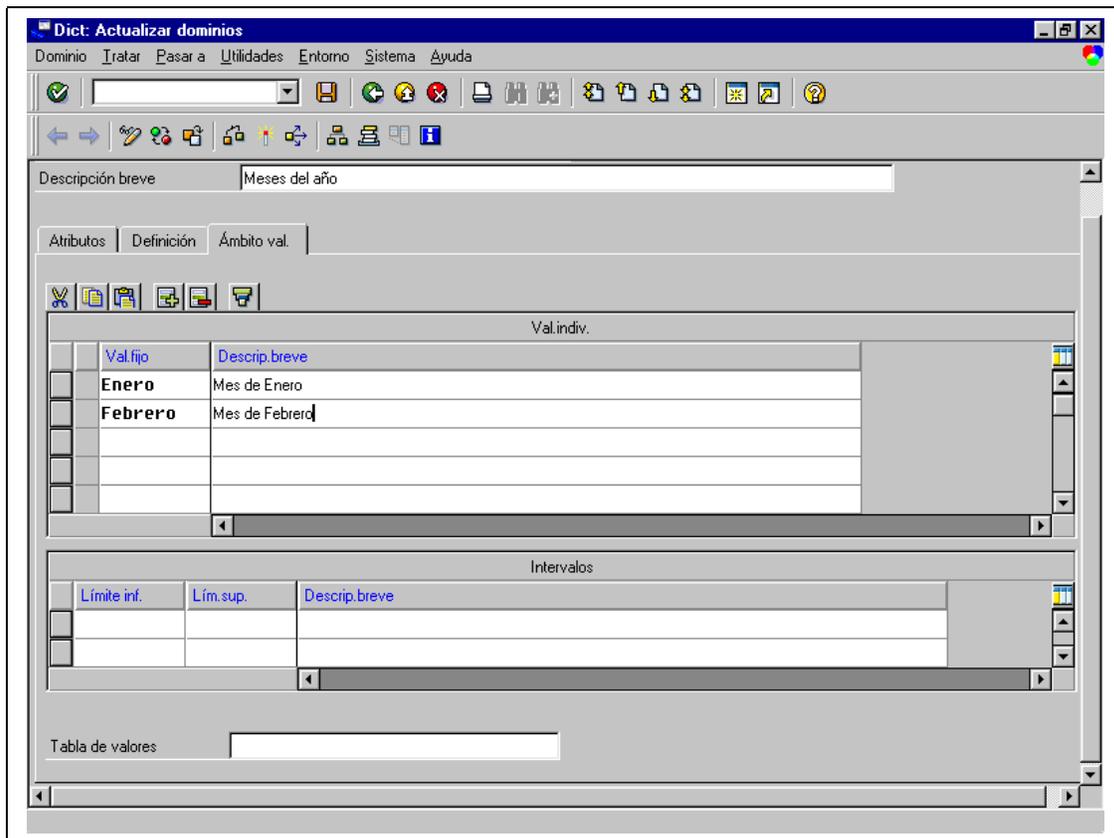
Rutina de conversión: Es una referencia a los procedimientos de conversión entre el formato interno del dato y su representación externa. *(En blanco)*

Flag de Minúsculas: Se permite la utilización de letras minúsculas *(No marcado).*

En la pestaña de *Ámbito Val:*

Representa el rango de datos válidos que puede tomar dicho atributo (El sistema realiza comprobaciones automáticas verificando que el valor introducido por pantalla está dentro de los valores válidos). Para definir estos valores (no siempre es necesario) hay dos posibilidades:

- Que estos valores estén almacenados en una tabla, en cuyo caso introduciremos aquí el nombre de la tabla.
- O bien fijar los valores directamente, bien valores individuales, bien rangos o intervalos de valores.



Como vemos, podemos introducir valores fijos junto con su descripción así como introducir intervalos de valores introduciendo el límite inferior y superior.

Una vez completados todos los campos deseados grabamos el dominio mediante la opción grabar  (F11)

Cuando grabemos por primera vez (no así en las modificaciones) nos preguntará por la clase de desarrollo a la que pertenece el objeto. La clase de desarrollo sirve para agrupar objetos pertenecientes a una misma aplicación y de esta manera poder transportar los objetos. Utilizaremos la clase de desarrollo '\$TMP' que es la correspondiente a los objetos locales y pulsamos el botón de grabar o bien pulsamos directamente el botón  que asignará directamente la clase de desarrollo temporal (Esto es extensible a todos los objetos posibles, dominios, elementos de datos, tablas, programas ...).

Una vez grabado el dominio, verificamos que no contiene errores para ello pulsamos el botón de verificar  (Ctrl. + F2), posteriormente será necesario activarlo para poder ser utilizado pulsaremos el botón de activar  (Ctrl. + F3). (Es necesario activar los objetos que se crean ya que hasta que no son activados no podrán ser utilizados en otros objetos).

La modificación de un dominio, se hace de manera similar, pondremos el nombre del dominio a modificar y pulsaremos la opción de modificar, una vez realizadas las modificaciones oportunas a las características del dominio grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl + F3).

(Crearemos los siguientes dominios:).

Dominio	Tipo	Long.	Descripción
<i>ZNCLIEXX</i>	<i>CHAR</i>	<i>10</i>	<i>Número de cliente</i>
<i>ZNOMBXX</i>	<i>CHAR</i>	<i>20</i>	<i>Nombre cliente</i>
<i>ZAPELLXX</i>	<i>CHAR</i>	<i>25</i>	<i>Apellidos</i>
<i>ZNFACTXX</i>	<i>CHAR</i>	<i>10</i>	<i>Número de factura</i>
<i>ZFECHAXX</i>	<i>DATS</i>	<i>8</i>	<i>Fecha</i>
<i>ZIMPNTXX</i>	<i>CURR</i>	<i>13</i>	<i>Importe</i>

3.3 Elementos de datos.

Si el dominio representa la parte técnica de un atributo, el elemento de datos representa la parte funcional del atributo, es decir, su descripción semántica.

Creación / Modificación / Visualización.

En el menú diccionario:

Seleccionamos la opción “Tipo de Datos”, introducimos el nombre del Elemento de datos que vamos a crear, (*en nuestro caso ZMESEFAXX*). Pulsamos la opción de Crear.

Seleccionamos la opción ‘Elem. Datos’, llegaremos a la siguiente pantalla:

The screenshot shows the SAP 'Dict: Actualizar elemento datos' dialog box. The 'Elemento datos' field contains 'ZMESEFAXX' and 'nuevo(revisado)'. The 'Descripción breve' field contains a question mark. The 'Tipo de datos' section has 'Dominio' selected. The 'Atributos' section has 'ID parámetro' and 'Componente por defecto' fields. The 'Ayuda para búsqueda' section has 'Nombre' and 'Parámetros' fields.

Descripción breve: Introduciremos una descripción representativa del objeto. (*‘Mes de la factura’*).

En la pestaña *Definición*:

Opción *Tipo elemental*:

Dominio: Dominio al que hace referencia el elemento de datos. (*ZMESEFAXX*).

Tipo Instalado: Tipo de datos y longitud con el mismo significado que el que se define en los dominios.

ID parámetro: Permite referenciar a un parámetro de memoria SAP. Será útil para mostrar valores por defecto en pantallas, ya que este campo se completará con el valor que tenga el parámetro de memoria SAP al mostrar la pantalla. (*En nuestro caso lo dejamos en blanco.*)

En la pestaña *Denom. Campo*: Estos campos corresponden a la descripción del objeto, estos, son los textos que se mostrarán en los diferentes lugares donde se hagan referencia al campo que utilicen el elemento de datos , por ejemplo en cabeceras de

informes al visualizar contenidos de tablas, en pantallas... El campo longitud representa el espacio en el que se va a escribir el texto.

(En nuestro caso: Mes Fac., Mes de factura, Mes de la factura.

En el campo longitud pondremos los valores propuestos: 10, 15, 20).

Una vez completado, grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl. + F3).

(Crearemos los siguientes elementos de datos :)

Elem. Datos	Dominio	Descripción
ZNCLIEXX	ZNCLIEXX	Número de cliente
ZNOMBXX	ZNOMBXX	Nombre cliente
ZAPEL1XX	ZAPELLXX	Primer Apellido
ZAPEL2XX	ZAPELLXX	Segundo Apellido
ZNFACTXX	ZNFACTXX	Número de factura
ZFECHAXX	ZFECHAXX	Fecha factura
ZIMPNTXX	ZIMPNTXX	Importe neto

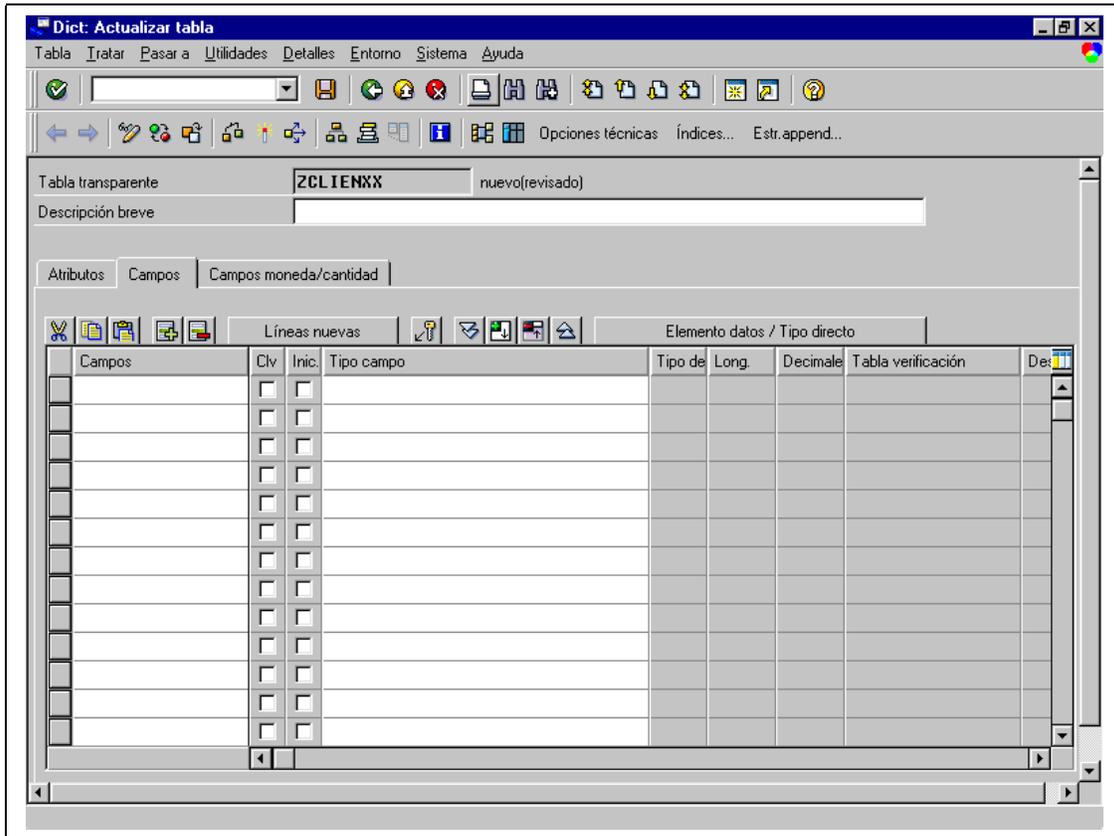
3.4 Tablas.

3.4.1 Creación de una tabla.

Una tabla representa un conjunto de atributos de una entidad. Esta formada por campos que se corresponden con cada uno de los atributos. Cada uno de ellos hará referencia a un elemento de datos.

Creación / Modificación / Visualización.

En el menú diccionario introducimos el nombre de la tabla que vamos a crear, en nuestro caso ZCLIENXX, seleccionamos la opción "Tablas" y pulsamos la opción de Crear.



Descripción breve: Descripción significativa de la tabla. (*Maestro de clientes*)

Pestaña *Atributos*:

Clase de entrega: Indica quién es el responsable del mantenimiento de la tabla, si es una tabla de parametrización,... (*Pondremos de tipo A (Aplicación)*).

Permitida Actualización tabla: Habilita/ Deshabilita la posibilidad de que el contenido de la tabla pueda ser modificado en la transacción de visualización del contenido de la tabla. (Si no se marca no se podrán modificar los registros de la tabla por esta transacción). (*Marcamos con una X*)

Pestaña *Campos*:

Nombre campo: Nombre del campo.

Clave: Indica si el campo forma parte de la clave primaria de la tabla. Un campo o conjunto de campos son clave en una tabla si determinan de forma unívoca un único registro de dicha tabla. Toda tabla tiene que tener clave primaria.

Tipo de datos: Nombre del elemento de datos que describe al campo. (*)

Tipo y longitud: Tipo de dato y longitud. (*)

Tabla de Verificac.: En esta tabla aparecerá un "*" cuando el elemento de datos introducido, haga referencia a un dominio que tenga una tabla de valores permitidos, o bien cuando se asocie una clave externa (Visto más adelante).

Descripción breve: Descripción del campo. (*)

(*) Podemos crear campos en la tabla que no necesariamente han de estar vinculados a un elemento de datos, si no que directamente le podemos asignar un tipo, longitud y descripción. Mediante el botón  de esta manera se habilitan para entrada estos campos. Para volver a introducir tipos de datos pulsaremos nuevamente esta opción. (Aunque existe esta posibilidad, no es muy recomendable).

Pestaña *Campos moneda/cantidad*:

TabRef y Cpo Ref.: Solamente los tipos de datos de importe (CURR) y cantidad (QUAN), necesitan ser referenciados a otros campos del diccionario. Para ellos es obligatorio completar estos valores (tabla + campo). Estos campos de referencia deberán de ser del tipo Moneda (CUKY) para importes y Unidad (UNIT) para cantidades. De esta forma, cuando se visualicen datos (en pantallas, pantallas de selección...) serán formateados con el valor que contenga el campo al que han sido referenciados.

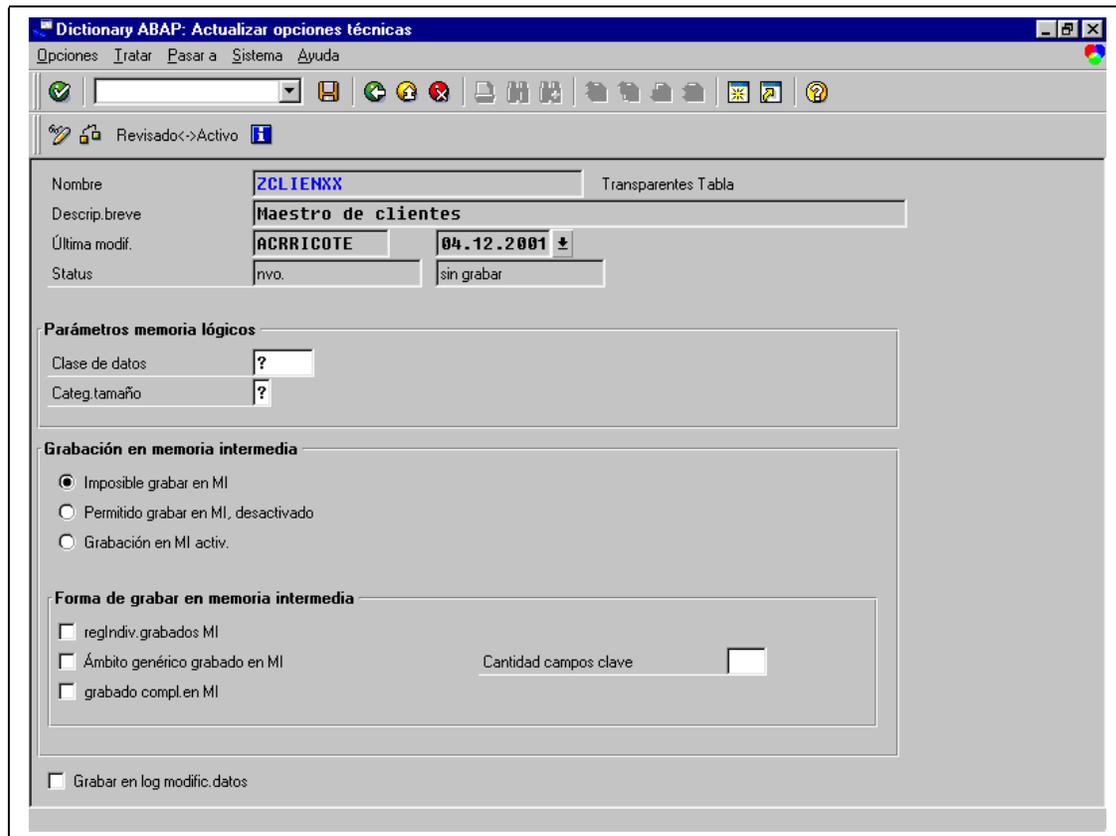
Añadimos a la tabla los campos:

Campo	Cl.	Elem. Datos	Tipo	Long	Descripción
MANDT	X	MANDT			Mandante
BUKRS	X	BUKRS			Sociedad
NCLIE	X	ZNCLIEXX			Nº de identificación cliente
NOMBR		ZNOMBXX			Nombre cliente
APEL1		ZAPEL1XX			Primer apellido
APEL2		ZAPEL2XX			Segundo apellido
FNACI			DATS	8	Fecha de nacimiento

Nota: El campo MANDT, mandante, se añade como un atributo en las tablas de esta forma, se pueden tener varias colecciones de datos distintas según el mandante. En las selecciones, actualizaciones... de la tabla este campo es transparente ya que tendrá siempre el valor indicado al iniciar la sesión en el sistema. (No es obligatorio definir el mandante en todas las tablas aunque si es lo más habitual).

OPCIONES TÉCNICAS

Una vez completados todos los campos, grabamos la tabla  (F11). Posteriormente será necesario completar las definiciones técnicas de la tabla para ello seleccionaremos la opción de menú 'Pasar a → Opciones técnicas' o el botón . Aparecerá la siguiente pantalla, donde hay que completar los campos:



Clase Datos: Con la clase de datos se define de forma lógica el ámbito físico en la base de datos en el que se grabará la tabla. Este lugar físico donde se ubicará la tabla se determinará en función del tipo de utilización de la tabla, es decir, predominio de actualizaciones ó consultas.... Por ejemplo una tabla con datos maestros sufrirá pocas actualizaciones y muchas consultas... *(Para nuestro ejemplo seleccionaremos APPL0)*

Categoría tamaño: Determina el número aproximado de registros que va a albergar la tabla. *(Seleccionamos tamaño 3)*

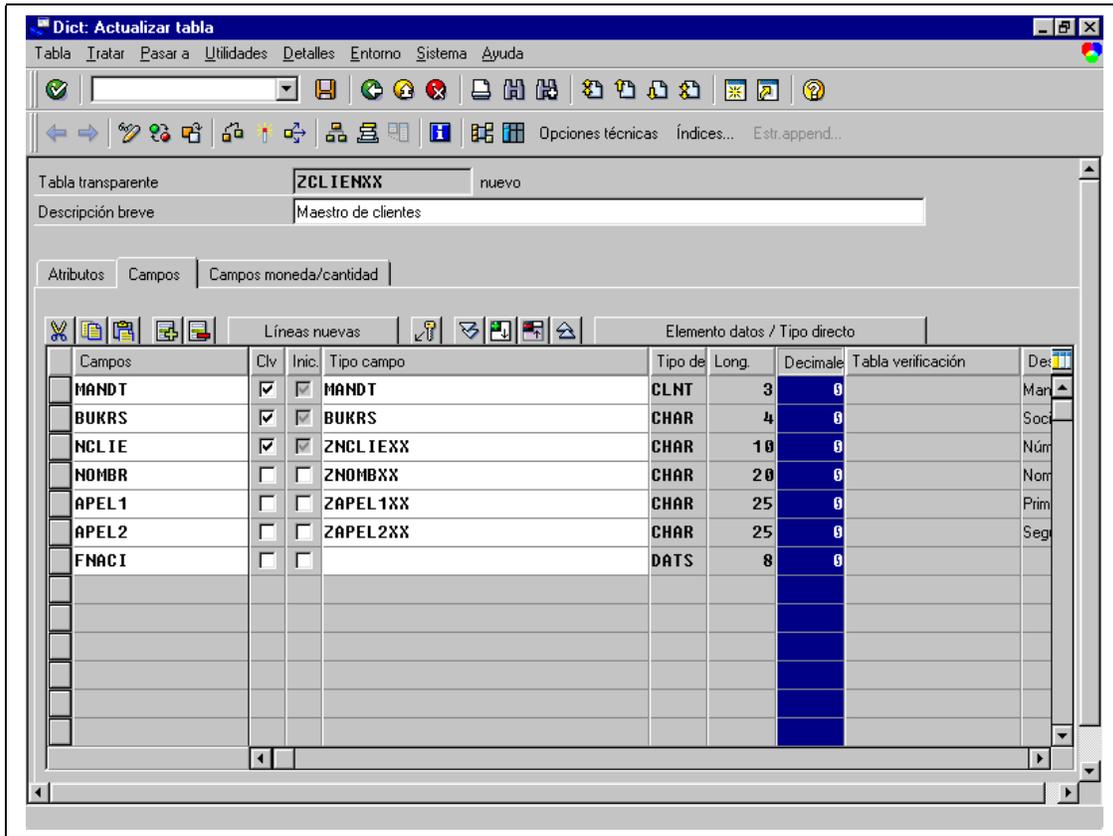
Una vez completados, pulsamos el botón verificar  (Ctrl. + F2) y grabamos botón  (F11). Volveremos a la pantalla inicial con el botón  (F3) verificamos  (Ctrl. + F2) y activamos la tabla  (Ctrl. + F3).

3.4.2 Índices.

Un índice es la ordenación de los registros de una tabla, por uno o varios de los campos que la componen. Por tanto la definición de un índice consisten en la enumeración de los campos que lo componen.

Para crear un índice a una tabla, en la pantalla:

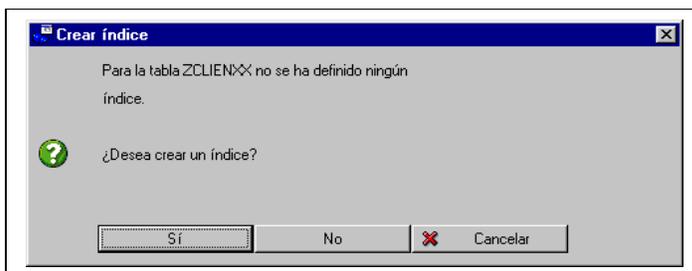
Curso programación ABAP IV



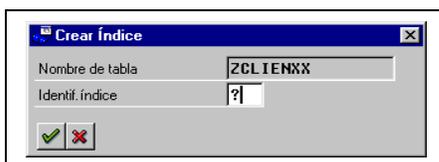
Pulsamos la opción de menú 'Pasar a → índices' o el botón

Índices...

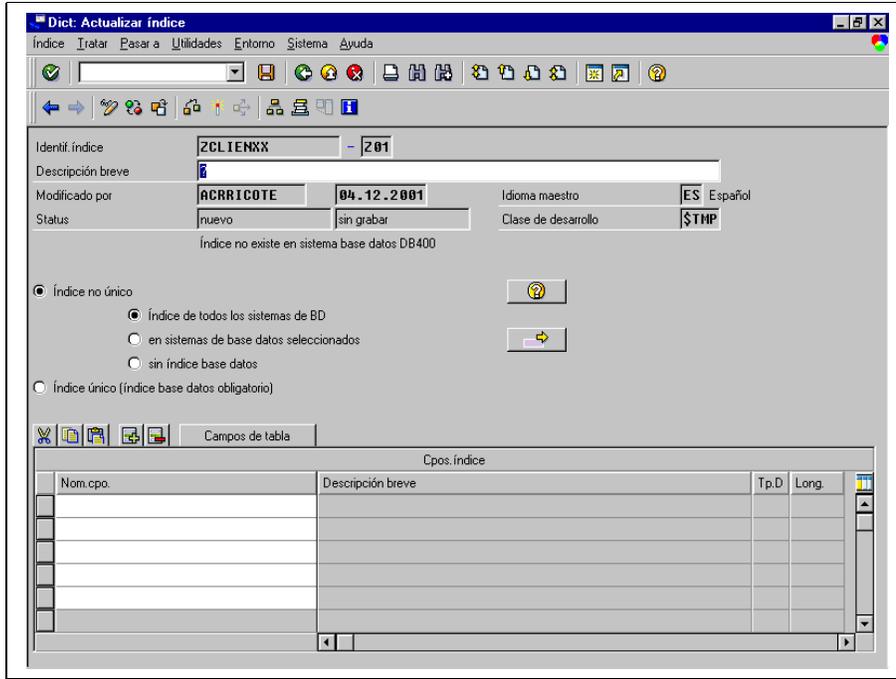
Si no hay ningún índice creado para la tabla aparecerá la siguiente ventana:



Al pulsar la opción 'Sí' aparecerá la ventana donde nos pide un identificador para el índice. (Pondremos Z01).



A continuación aparece la pantalla donde se introducen los campos que forman en índice.



Descripción breve: Descripción del índice. (*Índice por fecha de nacimiento.*)

Índice único / no único: Si marcamos la opción índice único haremos que el índice sea único, es decir que no se podrán repetir entradas iguales en la tabla para los campos que formen el índice. En nuestro ejemplo si marcamos el índice como único no se podrán insertar en la tabla dos clientes que nacieran el mismo día. (*Marcamos el índice como no único.*)

Campos: Campos que forman el índice. Pondremos el nombre de los campos o bien podemos seleccionarlos mediante el botón . (*En nuestro caso seleccionaremos los campos MANDT y FNACI.*)

Cuando se han completado los datos grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl. + F3).

Para modificar el índice pulsaremos el botón  nos mostrará una ventana con los índices existentes :

Nomb	Unique	Descripción breve	Status
201	<input type="checkbox"/>	Índice por fecha de nacimiento	activo

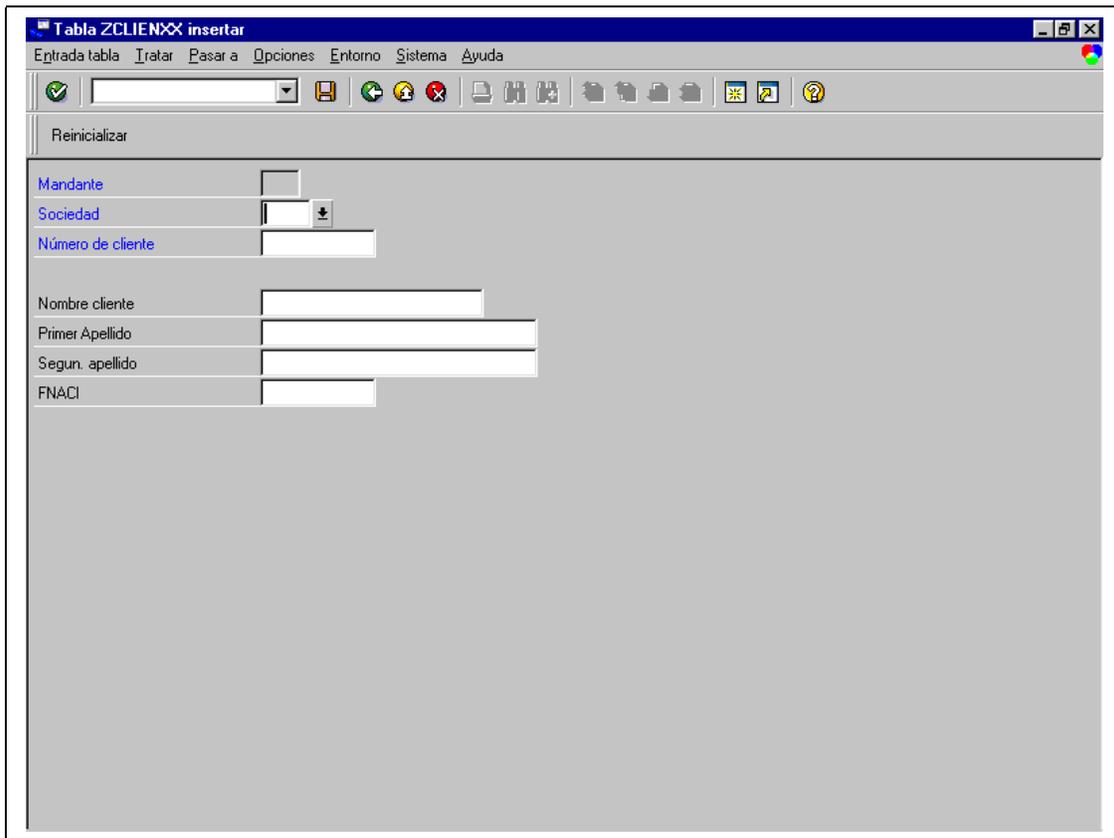
Podremos seleccionar uno de los existentes para modificarlo o crear uno nuevo 

En general los índices permiten acelerar las operaciones de consulta, pero en contraposición ralentizan las actualizaciones ya que cada actualización de alguno de los registros de la tabla ha de actualizar el/los índices de la misma. Por tanto a la hora de crear índices hay que valorar bien su utilidad y repercusiones.

Toda tabla está ordenada por los campos clave, luego hay un 'índice' implícito por estos campos, y no tendrá sentido definir un índice con los todos los campos clave. (En el ejemplo sería absurdo definir un índice con los campo MANDT, BUKRS y NCLIE).

3.4.3 Visualizar / Modificar Contenido de tabla.

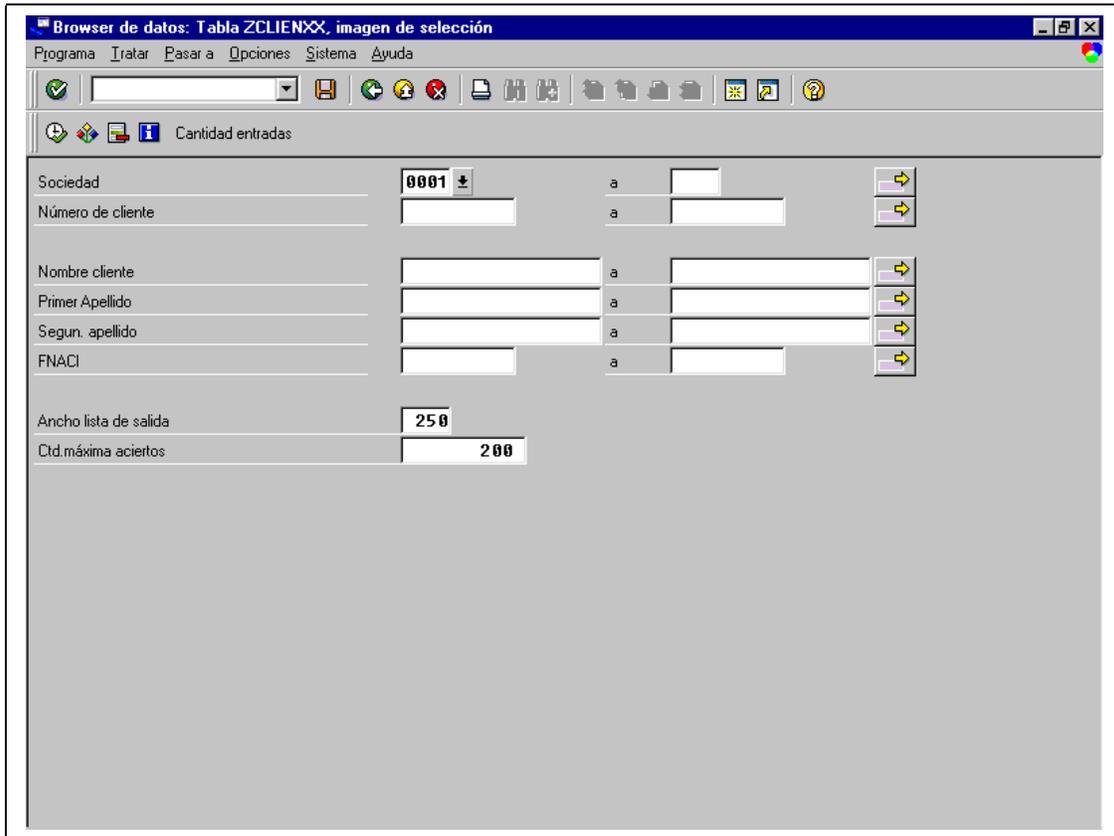
Una vez que la tabla está definida, podemos almacenar información para ello dentro de la pantalla de definición de la tabla seleccionamos la opción de menú 'Utilidades →Contenido tabla→Registrar entradas' aparecerá una pantalla en la que introducir cada uno de los registros:



Introducimos los datos y pulsamos  (F11).

Para visualizar los registros de la tabla seleccionaremos la opción de menú 'Utilidades→Contenido tabla→ Visualizar'.

Aparecera una pantalla de selección de la tabla donde se podrán indicar los criterios de búsqueda.



Pulsando  visualizaremos el contenido de la tabla en forma de listado.

The screenshot shows the SAP Data Browser interface for table ZCLIENXX. The window title is 'Data Browser: Tabla ZCLIENXX' and it shows '1 aciertos'. The menu bar includes 'Entrada en tabla', 'Tratar', 'Pasara a', 'Opciones', 'Utilidades', 'Entorno', 'Sistema', and 'Ayuda'. The toolbar contains various icons for navigation and actions. Below the toolbar, the table name 'Tabla: ZCLIENXX' and 'Campos visualiz.: 7 De 7 Columnas clave fijas: 3 AnchoLista 0250' are displayed. The table has the following structure:

X	Mandante	Sociedad	Número de cliente	Nombre cliente	Primer Apellido	Segun. apel
<input type="checkbox"/>	600	0001	0000000001	PEDRO	PÉREZ	LÓPEZ

Desde esta pantalla podremos, crear nuevas entradas  y visualizar  , modificar  y borrar ('Entrada en Tabla→ Borrar ') las existentes.

Existen otras formas de llegar a la visualización del contenido de una tabla sin pasar por la transacción del diccionario de datos.

Con la transacción **SE16**, indicaremos el nombre de la tabla y llegaremos directamente a la pantalla de selección de datos de la tabla.

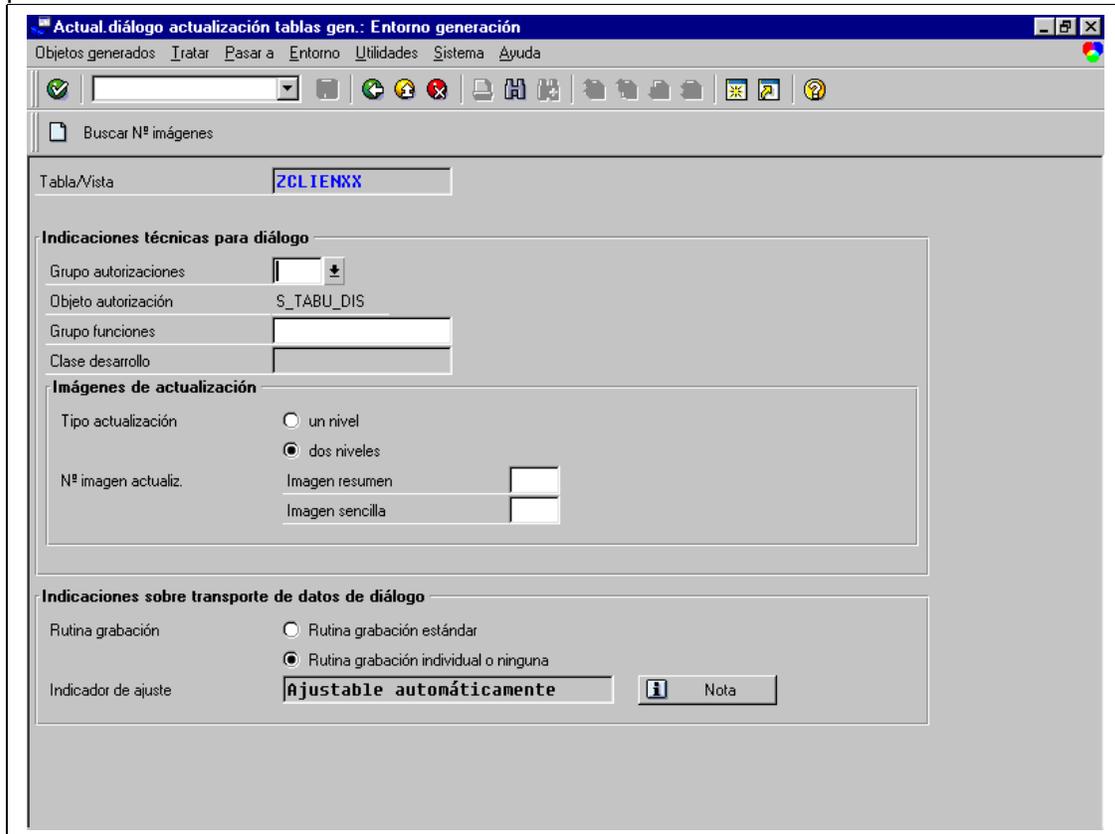
Con la transacción **SM30** (sólo si hemos generado las vistas de diálogo de actualización de la tabla).

3.4.4 Vistas / Diálogo de actualización. **GENERACION ACTUALIZACION DE TABLAS**

Un diálogo de actualización de una tabla es un conjunto de programas cuya utilidad es el mantenimiento del contenido de la tabla. Cuando creamos el diálogo sobre una tabla, se generan estos programas de forma automática.

Aunque no es estrictamente necesario crear este diálogo para mantener las tablas si es muy aconsejable para aquellas tablas que el usuario mantiene ya que de esta forma se puede controlar de manera sencilla la seguridad de la tabla. (Esta opción es muy utilizada en las tablas de parametrización, por ejemplo).

Para generar el diálogo, la tabla ha de estar activada. Una vez activada seleccionamos la opción de menú 'Utilidades → Generador Actualiz.Tab.' llegaremos a la siguiente pantalla:

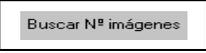


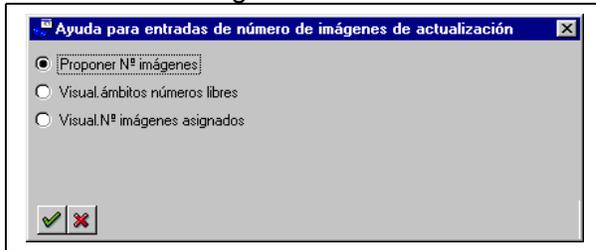
Grupo de autorizaciones: Mediante este campo, se puede controlar la seguridad sobre el mantenimiento de la tabla, es decir, limitará los usuarios que puedan modificar el contenido de la tabla. Solamente podrán realizar modificaciones aquellos usuarios que tengan autorización sobre el grupo de autorizaciones indicado. Si deseamos no controlar la autorización pondremos en este campo el valor por defecto '&NC&'. (En nuestro caso pondremos este valor).

Grupo de funciones: Es necesario indicar un grupo de funciones en el que se incluirán las funciones, dynpros y demás objetos del diálogo de actualización. Aunque podemos indicar aquí cualquier grupo de funciones ya existente, es conveniente indicar uno nuevo o bien un grupo que contenga una funcionalidad similar, el diálogo de actualización de otra tabla, por ejemplo. (En nuestro caso pondremos ZGXX).

Tipo de actualización: Marcar uno o dos niveles hará que se generen una o dos pantallas de mantenimiento. Si marcamos un nivel solamente se generará una pantalla (imagen resumen) que muestra el contenido de todos los registros de la pantalla, si dos niveles además de la pantalla anterior (imagen resumen) se genera una segunda (imagen sencilla) con el detalle de cada uno de los registros. (Seleccionamos dos niveles).

Imágenes actualización: Representan el número de dynpro que se asignará a cada una de las dos pantallas que se generan. Podemos indicar dos números de dynpro que

no existan en grupo de funciones elegido aunque existe una utilidad para que el sistema elija de manera automática estos valores para ello pulsamos el botón  nos llevará a la siguiente ventana



Con la primera opción nos propondrá unos números no existentes. Las otras opciones permiten visualizar los nº todavía libres y los ya asignados. Al seleccionar la primera opción volveremos a la pantalla anterior, donde se habrán informado los campos de la imagen resumen y sencilla. Pulsamos el botón crear  (Si el grupo de funciones indicado no esta creado, nos pedirá una clase de desarrollo... *pondremos seleccionaremos objeto local*). Grabamos  (F11) y retornamos a la pantalla anterior  (F3).

Para introducir entradas y visualizar el contenido de la tabla seguiremos los mismos pasos que en el apartado anterior. (Veremos como las pantallas han cambiado ya que ahora se verán las pantallas de diálogo generadas)

3.4.5 Claves Externas.

Un campo de una tabla tiene asociada una clave externa cuando los valores posibles que puede tomar el campo en dicha tabla, han de estar en otra tabla/s que llamaremos tabla maestra.

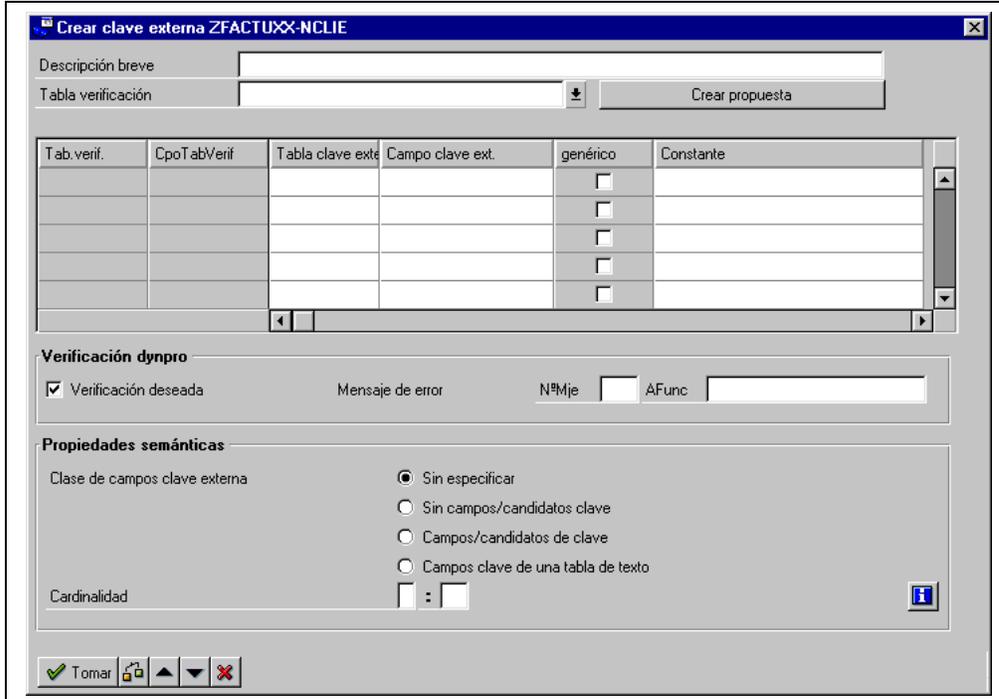
Vamos a crear la siguiente tabla teniendo en cuenta las indicaciones que se detallan a continuación para los campos NCLIE e IMPNT

Nombre tabla: ZFACTUXX:

Campo	Cl.	Elem. Datos	Tipo	Long	Descripción
MANDT	X	MANDT			Mandante
BUKRS	X	BUKRS			Sociedad
NFACT	X	ZNFACT XX			Nº de factura
NCLIE		ZNCLIE XX			Número de cliente
FECHA		ZFECHAXX			Fecha de la factura
MESFA		ZMESFAXX			Mes de la factura
IMPNT		ZIMPNTXX			Importe neto de la factura
MONED		WAERS			Clave de moneda

En esta tabla el campo NCLIE tendrá unos valores limitados a los existentes en la tabla maestro de clientes. (No deberíamos tener facturas de clientes que no existen) Para

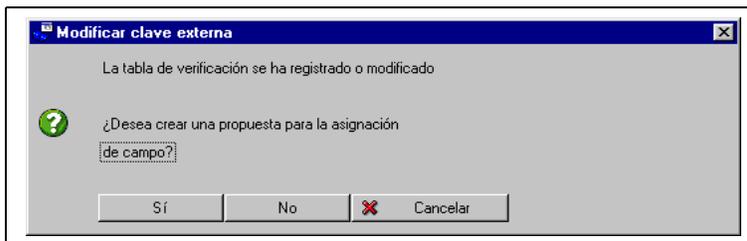
asociar esta clave externa, situados sobre este campo seleccionamos la opción de menú 'Pasar a → Claves Externas' o botón  nos aparecerá la ventana:



Descripción Breve: Descripción de la clave externa. (Pondremos 'Maestro de clientes').

Tabla de verificación: Tabla maestra que contiene el campo con los valores válidos (Pondremos ZCLIENXX).

Una vez indicada la tabla, nos aparecerá la ventana:



Seleccionamos la opción 'Sí' y no la pantalla se completará con los campos claves de la tabla introducida :

Cardinalidad: La cardinalidad entre dos tablas, representa el grado de la relación entre las mismas, es decir, cuantas ocurrencias de una tabla están relacionadas con una entrada de la otra. Por ejemplo en la relación entre madres e hijos la cardinalidad será 1 a N ya que una madre puede tener cero ó más hijos pero un hijo tendrá una y solamente una madre. Por tanto introduciremos aquí los valores oportunos en función de la relación entre las tablas. *(En nuestro caso seleccionaremos 1: CN ya que una un cliente puede tener varias facturas, pero una factura únicamente puede pertenecer a un cliente).*

Una vez introducidos lo valores oportunos verificamos  (Ctrl. + F2) validamos . Podemos ver como se indica el valor de la tabla en la columna de tabla de verificación.

En esta tabla tenemos un campo IMPNT que es de tipo importe (CURR) por tanto es necesario referenciarlo. Para ello podemos seleccionar la pestaña 'Campos moneda / Cantidad) e informar directamente los campo TabRef y CpoRef. o bien situados sobre el campo hacemos Doble-Click (F2) aparecerá la ventana donde también podemos indicar estos datos

Tab. Referencia: Nombre de la tabla. *(Pondremos ZFACTUXX).*

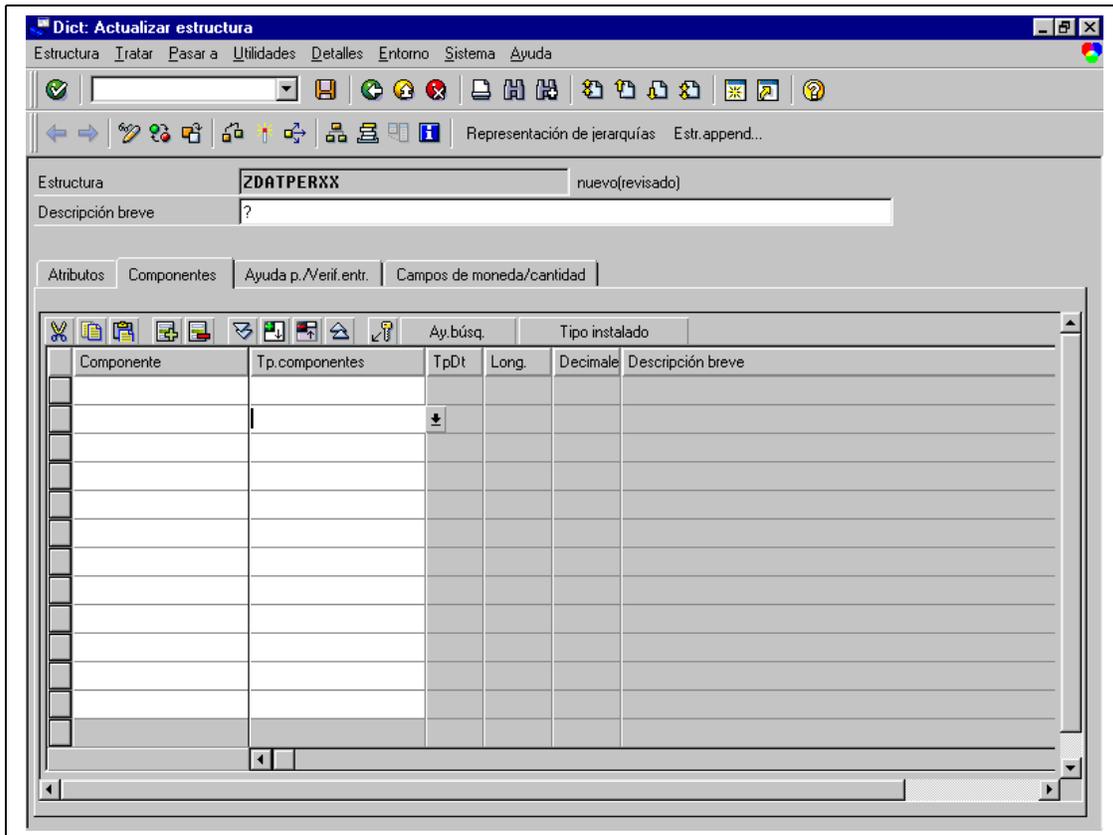
Cpo. Referencia: Campo de la tabla indicada. *(Pondremos Moned).*

De esta manera hemos referenciado el importe IMPNT a la moneda MONED.

3.5 Estructuras.

Una estructura es similar a una tabla en cuanto a su definición pero con la diferencia que una estructura no puede albergar registros. Una estructura es simplemente la definición de una entidad.

Para crear una estructura, desde la pantalla de diccionario, seleccionamos la opción 'Tipo de datos' ponemos el nombre de la estructura a crear (*En nuestro caso 'ZDATPERXX'*) y pulsamos crear, seleccionamos la opción 'Estructura'. Llegaremos a la pantalla:



Donde introduciremos los campos y su definición, bien a través del campo elemento de datos o bien introduciendo el tipo, longitud y descripción (Del mismo modo que en la creación de tablas). (*Introduciremos, los campos Nombr, Apel1, Apel2, con sus respectivos elementos de datos ZNOMBRXX, ZAPELLXX. En el texto pondremos 'Datos personales'*).

3.6 Vistas.

Una vista es una particularización de una o varias tablas en la que se pueden seleccionar uno o varios de los campos de las tabla/s.... Por ejemplo podemos realizar una vista para seleccionar únicamente algunos campos de una tabla pero principalmente utilizaremos vistas para obtener datos de tablas relacionadas. Bajo una

vista podemos 'unir' dos o más tablas y poder ver la intersección entre dichas tablas como si se tratará de una única tabla.

Cuando se trata de una vista de más de una tabla, es necesario definir las condiciones por las que se unen las tablas, es decir, los campos de intersección de las tablas. Las vistas no contienen datos sino que los datos se obtienen en tiempo de ejecución al realizar consultas sobre ellas.

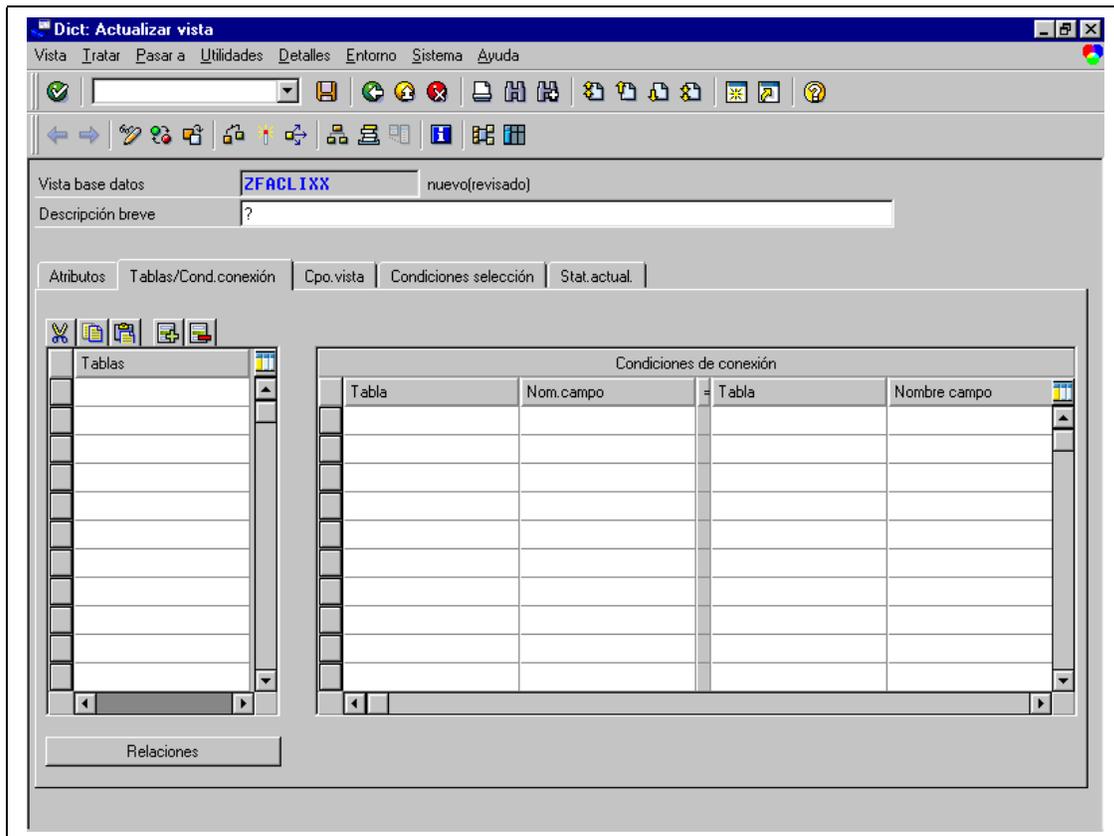
Por ejemplo: Si quisiéramos ver todas las facturas de un cliente (Nombre y primer apellido), deberemos realizar una intersección entre la tabla de clientes y la tabla de facturas para obtener una vista con los campos:

Facturas/ Cliente.

Campo	Descripción
<i>NFACT</i>	<i>Nº de factura</i>
<i>NCLIE</i>	<i>Nº de cliente</i>
<i>NOMBR</i>	<i>Nombre cliente</i>
<i>APEL1</i>	<i>Primer apellido</i>
<i>IMPNT</i>	<i>Importe Neto</i>
<i>MONEDA</i>	<i>Moneda</i>

Crear/Modificar/Visualizar.

En la pantalla principal del diccionario de datos ponemos el nombre de la vista a Crear ('ZFACLIXX') seleccionamos 'Vistas' y pulsamos el botón crear. Seleccionaremos el tipo 'Vista de base de datos'. Llegaremos a la pantalla:



Tablas: Seleccionaremos las tablas de las que se desee obtener información. (En nuestro caso la tablas ZCLIENXX y ZFACTUXX).

Condiciones Join: Definen los campos por los que se realiza la intersección entre las tablas seleccionadas. Para indicar las relaciones podemos utilizar el botón  o escribir directamente las relaciones.

(En nuestro caso:
 ZCLIENXX-MANDT = ZFACTUXX-MNADT
 ZCLIENXX-BUKRS = ZFACTUXX-BUKRS)
 ZCLIENXX-CLIEN = ZFACTUXX-CLIEN).

Campos Vista: Se seleccionan los campos deseados. Podemos renombrar el campo en la vista será el que pongamos en Campos Vista'. (Por defecto será el nombre original del campo seleccionado). (En nuestro caso NFACT, NCLIE, IMPNT y MONEDA de la tabla ZFACTUXX y NOMBR y APEL1 de la tabla ZCLIENXX).

Para facilitar la selección de campos de una tabla, situados sobre la tabla pulsamos el botón  donde marcaremos los campos deseados , una vez seleccionados aparecerán automáticamente en la pantalla. Una vez completada la vista grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl. + F3).

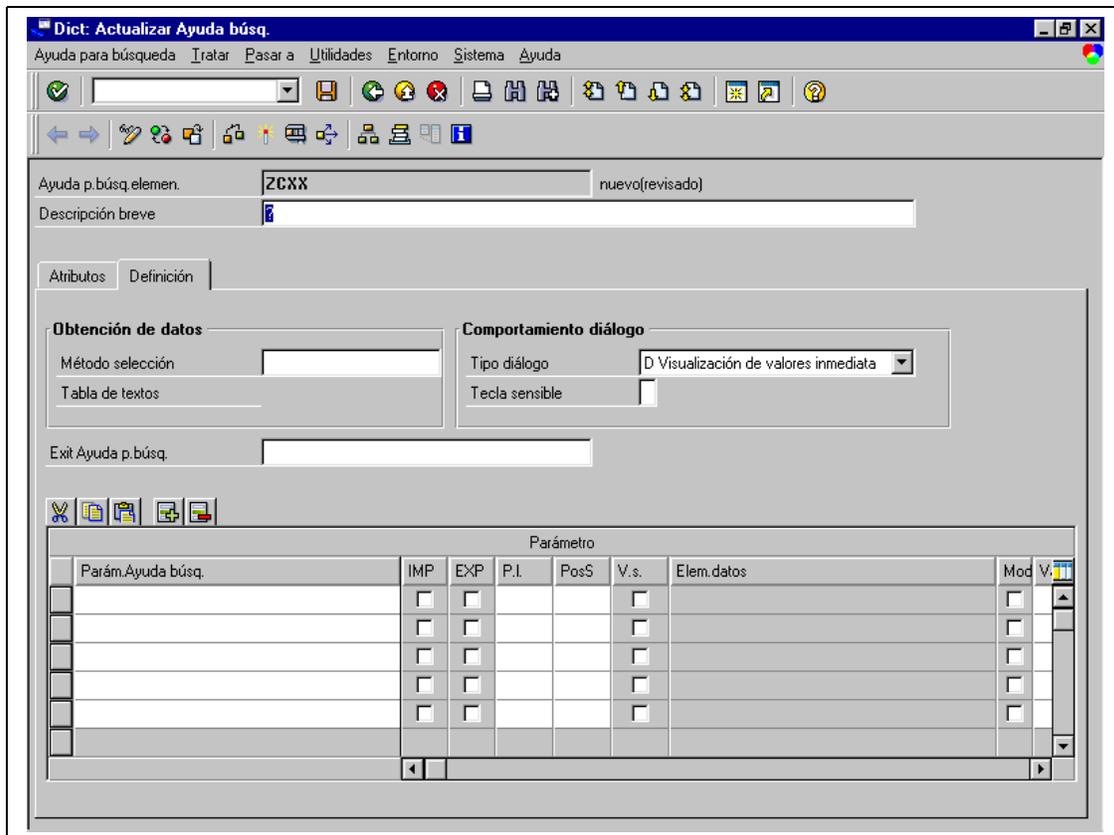
Para ver el contenido de la vista seleccionamos el menu 'Utilidades →Contenido'. Aparecera la pantalla de seleccón de la vista de la misma forma que en la visualización de los datos de una tabla.

3.7 Matchcode. (Ayudas para búsqueda)

Un matchcode es un instrumento de búsqueda de registros almacenados en el sistema. Permite en pantallas de seleccón... filtrar la información existente por determinados criterios para facilitar la búsqueda. Por ejemplo no sabemos el DNI de una persona pero si su nombre, mediante este, podremos obtener el DNI.

Crear/modificar/Visualizar:

En la pantalla principal del diccionario seleccionamos la opción Matchcode (Ayudas de búsqueda) introducimos el nombre del matchcode a crear, (ZCXX), y pulsamos el botón de crear, seleccionamos el tipo para ayuda elemental, llegaremos a la pantalla:



Descripción breve: Descripción del matchcode (*Podremos 'Búsqueda de clientes'*).

Método de selección: Nombre de tabla o vista del diccionario de datos de donde se obtendrán los datos a mostrar en la ayuda. (*Pondremos ZCLIENXX*).

Parám.Ayuda busq: Formado por cada uno de los campos que se utilizarán para la búsqueda.

IMP: Flag de parámetro EXPORT, este campo se utilizará como campo en el que informar valores de condición para la búsqueda de registros.

EXP: Flag de parámetro IMPORT, este campo se utilizará como parámetro de salida, es decir, se volcará el valor del registro seleccionado sobre el campo.

P.I: Posición del campo para lista de aciertos, indica la posición que ocupará el campo en la ventana de ayuda donde se mostrarán los registros seleccionados. (Si se indica '0' o Blanco, significará que el campo no se muestra en la lista de aciertos).

PoS: Posición del campo para la pantalla de condiciones de la selección.

V.s: Valor no modificable en la selección.

Valor Propuesta: Valor de propuesta para la selección.

(Seleccionaremos el campo NCLIE, NOMBR y APEL1 como campos de la búsqueda , seleccionaremos el NOMBR y APEL1 como campos para condición y NCLIE como parámetro EXPORT) .

A continuación grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl. + F3).

3.8 Objetos de bloqueo.

Sirven para controlar la concurrencia de procesos sobre un mismo objeto, siempre están asociados a tablas del diccionario. Un objeto de bloqueo es un semáforo sobre una tabla.

Cuando se define un objeto de bloqueo se generan automáticamente dos funciones que controlan dicho semáforo:

- ENQUEUE_Nombre: Controla la petición de bloqueo sobre el objeto.
- DEQUEUE_Nombre: Controla la liberación del bloqueo sobre el objeto.

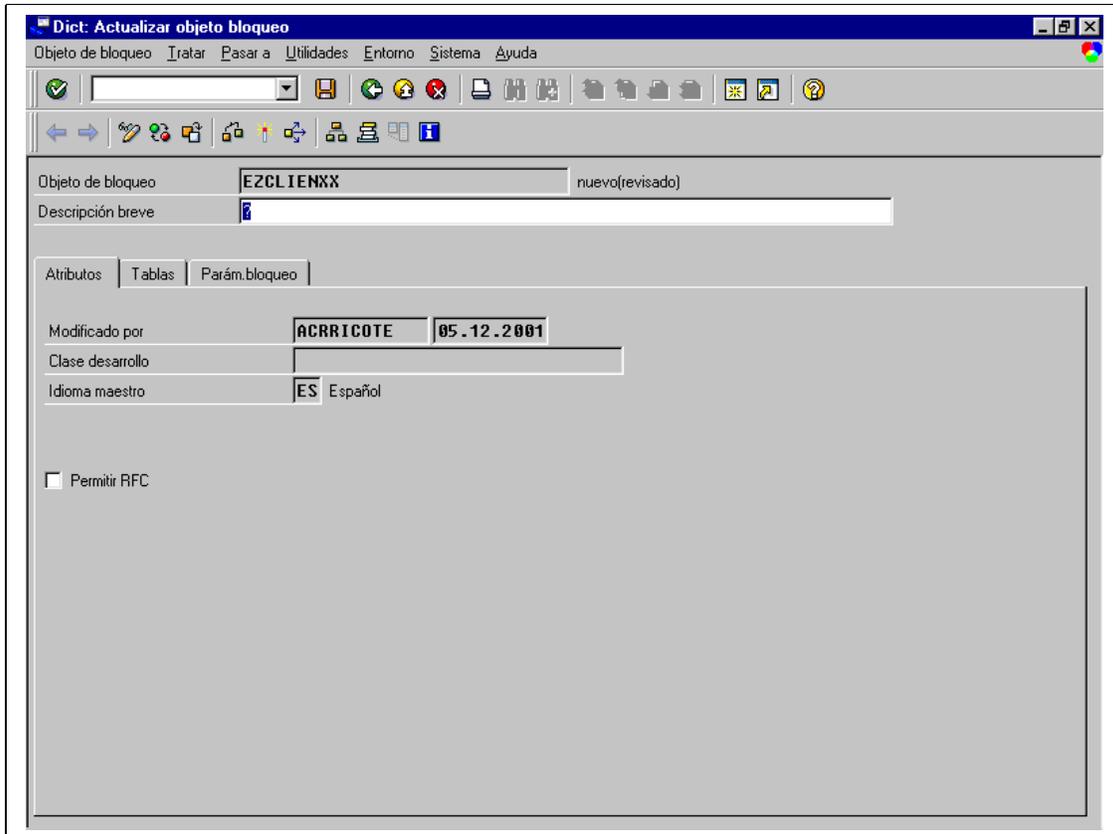
Los argumentos con los que se defina el objeto de bloqueo permitirán que se controle la totalidad de registros de la tabla, un conjunto de ellos ó un único registro.

Básicamente el funcionamiento es el siguiente:

Necesitamos que dos procesos no compartan un objeto de forma concurrente, por ejemplo que dos usuarios no accedan a la misma transacción simultáneamente, lo que haremos será programar un objeto de bloqueo, y al inicio de la transacción podremos la llamada a la función ENQUEUE_XXXX si el recurso esta ya siendo utilizado (encontramos semáforo rojo), no podremos bloquearlo y podremos obrar en consecuencia, mostrando un mensaje y terminar , mostrar una opción de reintento... Si se puede bloquear el objeto será señal que no hay otro proceso utilizándolo (semáforo verde) por tanto el proceso puede continuar (pondrá el semáforo en rojo para otros procesos), antes de terminar el proceso se deberá llamar a la función DEQUEUE_XXXX para liberar el objeto que se ha bloqueado (debe poner el semáforo en verde para que otros procesos puedan utilizarlo).

Para crear objetos de bloqueo, en la pantalla principal del diccionario de datos, seleccionamos la opción 'Objeto de bloqueo' *(En nuestro caso, vamos a crear un objeto*

de bloqueo para la tabla ZCLIENXX, escribiremos EZCLIENXX). Tras pulsar la opción de crear, nos aparecerá la pantalla:



Descripción breve: Descripción del objeto de bloqueo. (*Bloqueo maestro clientes*)

En la pestaña *Tablas*.

Nombre: Nombre de la tabla (*ZCLIENXX en nuestro caso*).

Modo de Bloqueo: Modo en el que se realiza el bloqueo, lectura, escritura, (*E*)

En la pestaña *Parám.bloqueo*.

Parám.bloqueo: Corresponden a los parámetros del argumento de bloqueo.

(*Dejamos los valores por defecto*)

Una vez completados todos los campos, grabamos  (F11), verificamos  (Ctrl. + F2) y activamos  (Ctrl + F3).

-

4. Programación de listados (REPORTS)

4.1 Introducción.

En el proceso de tratamiento de la información es necesario poder consultar los datos almacenados en el sistema de una forma organizada, para realizar este tipo de operaciones se utilizan programas tipo REPORT, con estos programas podemos generar listados en los que mostrar dicha información. Estos listados pueden estar orientados a visualización en pantalla y/o a impresora.

Podríamos distinguir dos tipos de listados por su grado de interacción con el usuario:

- **Reports planos:** Orientados a visualizar información de una manera plana en único nivel. En este tipo de listados las opciones de usuario sobre el tratamiento de la información es nula.
- **Reports interactivos:** Orientados a presentar información en varios niveles, es decir, pudiendo acceder desde un nivel de información a otro nivel de más detalle y así sucesivamente. Las opciones de usuario para el tratamiento de la información son amplias ya que se puede ir navegando a través de los distintos niveles de detalle.

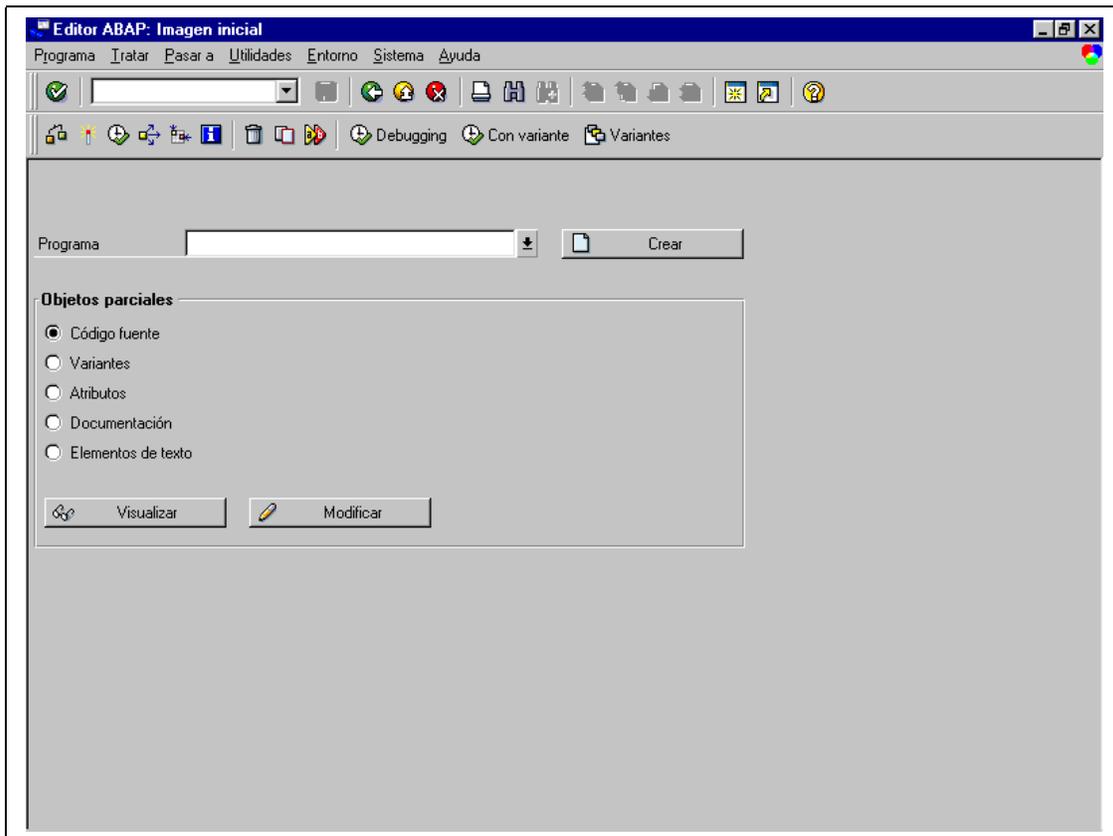
Por otra parte se da la necesidad de actualizar las bases de datos en las que no es necesaria la intervención directa del usuario, son los procesos que conocemos como 'procesos batch'. Estos programas también se realizan mediante programas tipo REPORT pero lógicamente sin la parte de presentación de datos por pantalla.

Por tanto con programas tipo REPORT podemos realizar listados para la visualización de información así como programas de actualización masiva de datos. Estos programas se realizan en un lenguaje de programación propio de SAP, es el lenguaje ABAP/4.

4.2 Editor ABAP/4

Para poder desarrollar los programas tenemos un editor de lenguaje ABAP/4. Este editor nos permitirá escribir, verificar y generar los programas. Hay varias maneras de llegar al editor, el camino más habitual para crear un nuevo programa del tipo REPORT es (desde el menú principal de SAP) Herramientas→ Workbench ABAP→Desarrollo→ Editor ABAP (SE38).

Llegaremos a la pantalla principal del editor de programas:



Desde esta pantalla podremos crear, visualizar, modificar y ejecutar (también podremos ejecutar en DEBUG mediante la opción **Debugging**, ya veremos su significado más adelante) los programas.

Para crear un programa, escribiremos el nombre del programa, seleccionaremos la opción 'Texto fuente' y seleccionaremos la opción de crear **Crear**. Nos aparecerá la siguiente pantalla donde se definen los atributos del programa.

The screenshot shows the 'ABAP: Propiedades de programa ZREPORTX modificar' dialog box. It contains the following fields and options:

- Título:** A text input field.
- Idioma maestro:** A dropdown menu with 'ES' selected, and 'Español' displayed next to it.
- Creado por:** A text input field with '05.12.2001' and 'ACRRICOTE'.
- Última modif.:** A text input field.
- Status:** A dropdown menu with 'Nuevo(revisado)' selected.
- Atributos:** A section containing:
 - Tipo:** A dropdown menu.
 - Status:** A dropdown menu.
 - Aplicación:** A dropdown menu.
 - Grupo autorizaciones:** A text input field.
- Bloqueo de editor:** An unchecked checkbox.
- Cálculo de coma fija:** A checked checkbox.

At the bottom, there is a toolbar with icons for 'Grabar', 'Cancelar', 'Ayuda', 'F11', and 'Cerrar'.

Título: Descripción de la funcionalidad del programa. (En nuestro caso pondremos 'Listado de maestro de clientes').

Tipo: Determina el tipo de programa, tenemos los siguientes:

Programa ON-LINE (1): Programa ejecutable tanto de forma ON-LINE y en fondo (BATCH)

Programa Include (I): Programa que contiene código pero que no puede ser ejecutado directamente si no que formará parte de otros programas ejecutable.

Module pool (M): Contiene código para controlar pantallas. Es necesario asociarle un código de transacción para que pueda ser ejecutado.

Grupo de funciones (F): Contiene programas denominados funciones, que no se pueden ejecutar directamente sino que tienen que ser llamados de otros programas.

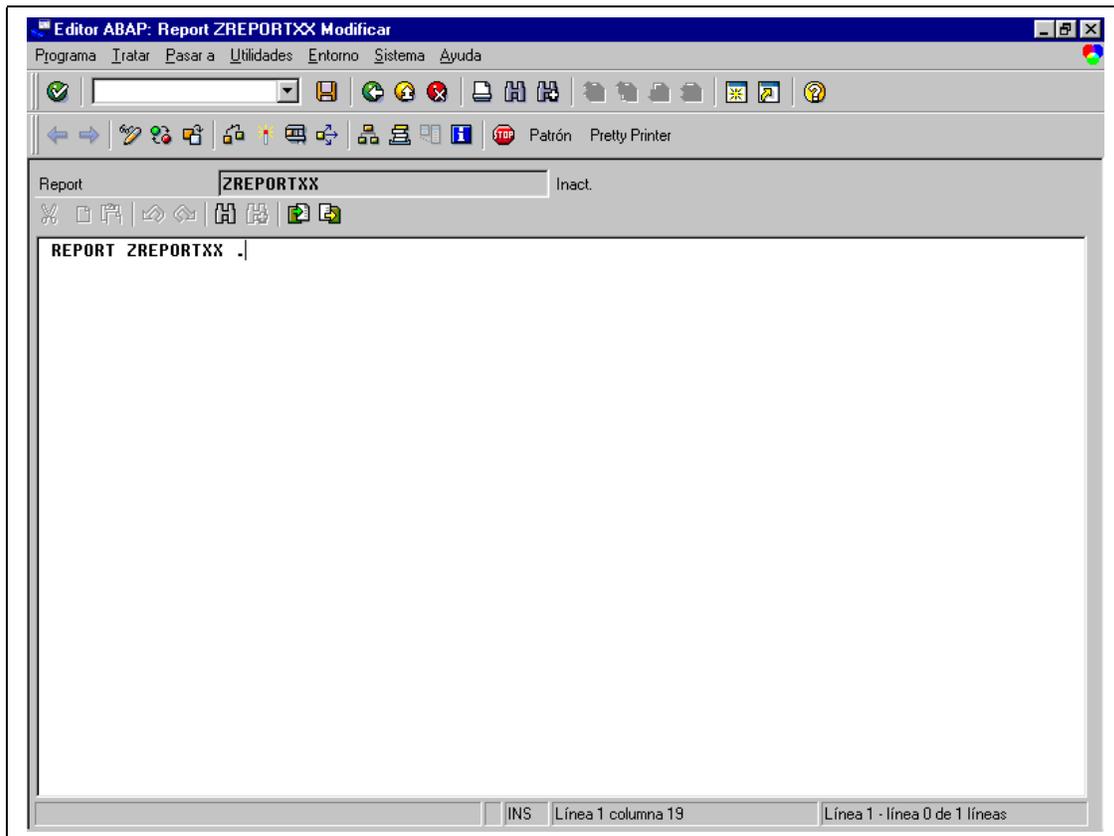
Pool de Subrutinas (S): Contiene código compuesto por procedimientos, tampoco puede ser ejecutado directamente sino que puede ser llamado desde otros programas.

(En nuestro caso pondremos tipo ON-LINE '1').

Aplicación: Módulo al que pertenece el programa (FI, HHRR...) (En nuestro caso pondremos un Aplicación Desconocida '**')

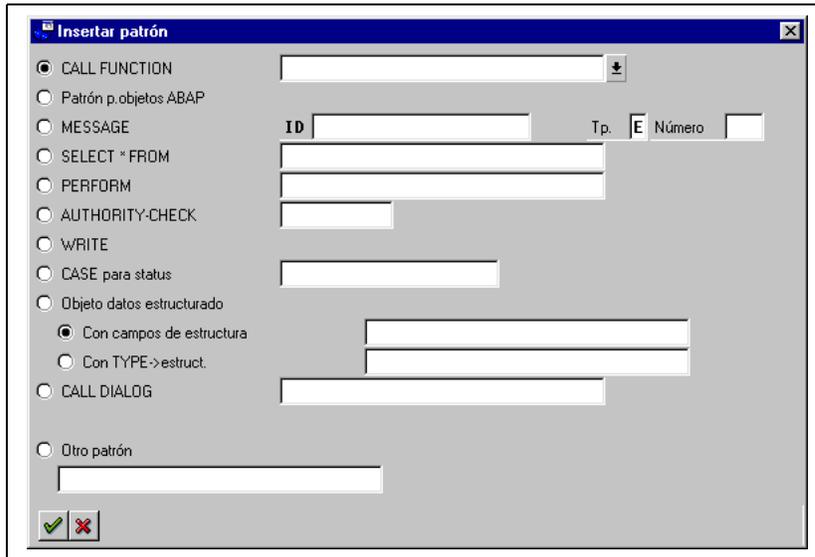
Una vez completado estos campos obligatorios pulsaremos el botón de grabar  (F11). Nos pedirá, como en los objetos de diccionario una clase de desarrollo,

grabaremos como objeto local botón  a continuación pasaremos a la pantalla del editor:



Vamos a ver las funciones básicas del editor para poder escribir, verificar, generar los programas.

Traer código patrón. Tenemos la posibilidad de insertar código con un patrón predefinido, para distintos elementos del lenguaje, sentencias como el WRITE, llamadas a procedimientos, llamadas a funciones ... para ello situados sobre la línea de código donde deseemos incluir esta llamada pulsaremos el botón nos aparecerá la siguiente ventana en la que indicaremos el nombre del objeto a insertar:



Obtención de ayuda sobre instrucciones...

Para obtener ayuda sobre una instrucción, pulsamos la tecla de ayuda 'F1'

Navegar a objeto.

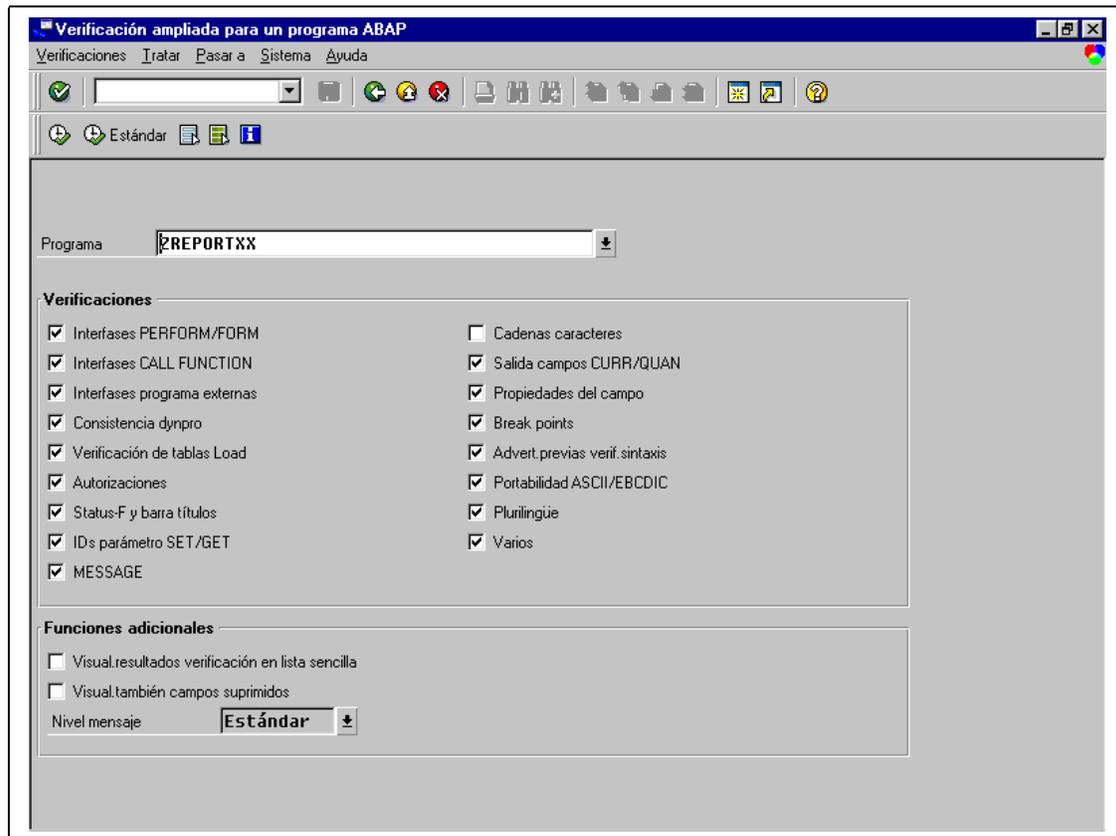
Esta función, permite 'navegar' hasta el objeto seleccionado, estos objetos pueden ser desde variables, procedimientos, símbolos de texto ... definidos dentro del propio programa hasta objetos definidos fuera del programa como pueden ser llamadas a transacciones, funciones, includes ...

Generalmente si el objeto al que pretendemos 'navegar' no existe, nos dará la posibilidad de crearlo en ese momento.

Verificación del programa.

Para comprobar que el programa no tiene errores de sintaxis utilizaremos el botón de verificar  (Ctrl. + F2).

Para verificar que no contiene errores de manera ampliada utilizaremos la opción de menú 'Programa→Verificar→Verif. programa ampliada'. Nos llevará a la siguiente pantalla, pulsaremos el botón  para continuar con la verificación.



Para generar el programa seleccionaremos la opción 'Programa→Generar' y lo activaremos con la opción 'Programa→ Activar' o  (Ctrl. + F3).

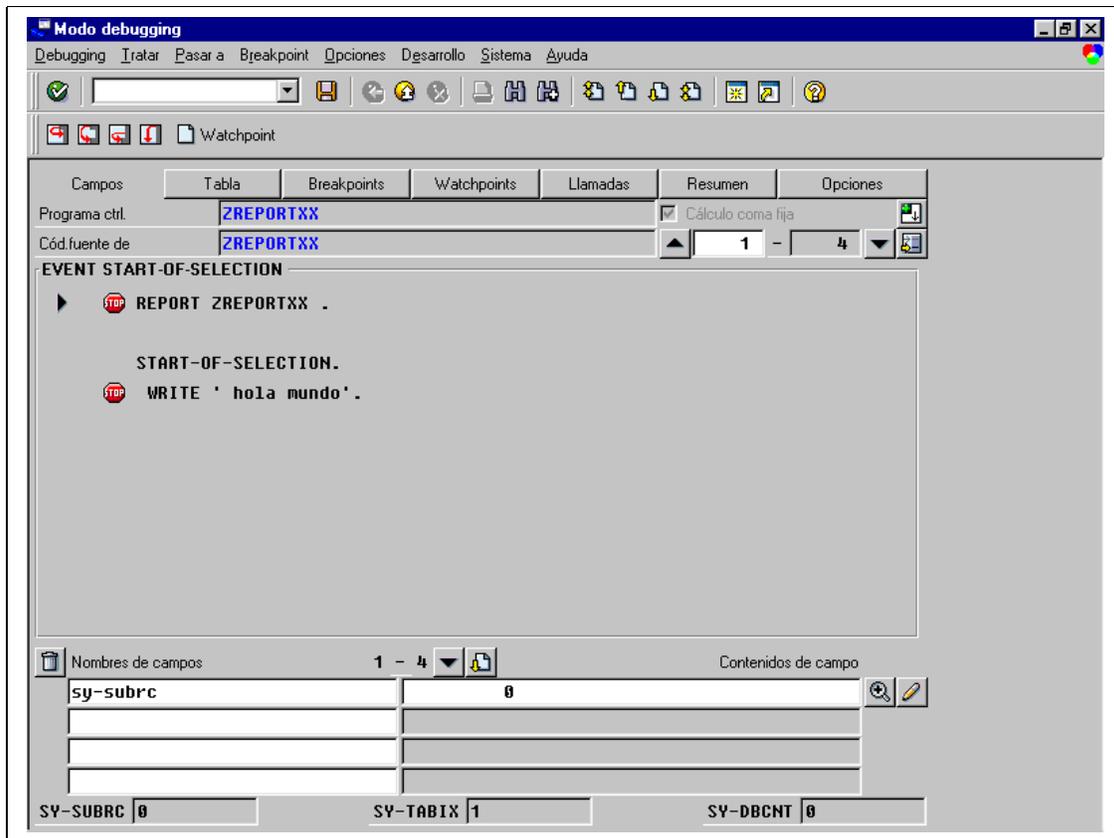
Ejecución del programa desde en editor.

Desde el editor, podemos ejecutar un programa. Para ello seleccionamos la opción de l menú 'Programa→Verificar' o  (F8).

Opción 'DEBUG'.

Para poder depurar los programas, podemos ejecutarlos en modo 'DEBUG', es decir, ejecutando el programa pero pasando por las líneas del código escrito y viendo el resultado de cada una de ellas de forma individual, de esta forma podemos comprobar el efecto de cada una de las instrucciones escritas. Para poder ejecutar un programa en modo 'DEBUG' desde el editor, será necesario fijar un 'Breakpoint', es decir un punto de ruptura. El punto de ruptura hace que la ejecución del programa se detenga cuando llega a la línea de código donde se encuentra y a partir de ese punto se puede ejecutar en modo 'DEBUG'. Para fijar un 'Breakpoint' nos situamos en la línea de código deseada y pulsamos el botón .

El aspecto de las pantallas del DEBUG es el siguiente:



El icono  indica que en esa línea de código hay un breakpoint.

El icono  indica la línea actual de programa (próxima instrucción a ejecutarse).

En la parte inferior (marco variables) podemos escribir el nombre de la variable de la que se quiere conocer el valor en ese momento, en la parte de la derecha aparecerá su valor (En la pantalla tenemos el nombre de la variable SY-SUBR y su valor actual '0'). Una vez dentro del 'DEBUG' de un programa tenemos las siguientes opciones:

Doble-click(F2) Situidos sobre una variable y pulsando este botón automáticamente se escribe en el marco variables para visualizar el valor actual de la variable.

 Hace que se ejecute la línea de programa actual. Si la línea actual se trata de la llamada a un procedimiento, función ... navegará hasta la primera línea ejecutable de dicho procedimiento.

 Ejecuta la línea actual de programa, teniendo en cuenta que si se trata de la llamada a un procedimiento ... no navegará hasta el código del procedimiento sino que se ejecutará el procedimiento como si se tratará de una única instrucción.

 Continúa con la ejecución de instrucciones hasta que se encuentre con el siguiente breakpoint. (Si no encuentra breakpoint continuará hasta el final).

 Retorna al nivel superior de anidamiento del programa, es decir, si hemos 'navegado' con la opción paso a paso dentro de un procedimiento, esta opción hará que se ejecute el resto del procedimiento y regrese a la instrucción siguiente a la llamada al procedimiento. (Si dentro del procedimiento existe algún breakpoint se parará en él).

 Permite visualizar el contenido de una tabla interna.

4.2.1 Documentación y estructuración de un listado plano.

Es fundamental que los programas estén bien estructurados y documentados, de esta manera haremos que su elaboración y el posterior mantenimiento sean mucho más sencillos. La estructura básica de un programa tipo REPORT es la siguiente (Esta estructuración puede variar dependiendo de los estándares particulares de cada cliente):

- Documentación básica del programa.
- Sentencia REPORT.
- Definición de tablas diccionario.
- Definición de constantes.
- Definición de variables.
- Definición de estructuras.
- Definición de tablas internas.
- Definición de rangos.
- Definición de la pantalla de selección:
- Parámetros
- Select-options
- Definición de FIELD-SYMBOLS.
- Definición de FIELD-GROUPS.
- Definición de Eventos:
- INICIALIZATION.
- START-OF-SELECTION.
- END-OF-SELECTION.
- TOP-OF-PAGE.
- END-OF-PAGE.
- AT LINE-SELECTION.
- AT USER-COMMAND.
- AT PFN.
- AT SELECTION-SCREEN.
- Includes.
- Subrutinas.

Antes de la primera sentencia REPORT, se deberá documentar el programa indicando:

- Nombre del programa.
- Descripción de la funcionalidad del programa.
- Fecha y autor del programa.

Deberá también reservarse un espacio para el control de modificaciones del programa.

(En nuestro ejemplo que va a consistir en programa que muestra un listado de los clientes existentes, con el siguiente formato)

C. Cliente Nombre Primer Apellido Segundo Apellido F.Nacimiento
X (10) X (20) X (25) X (25) XX (10)

El programa dará la posibilidad de seleccionar clientes por Código de cliente, por nombre y por fecha de nacimiento de tal forma que solamente aparecerán en el listado aquellos clientes que cumplan los criterios de selección indicados.

Pondremos como documentación en la cabecera del programa:

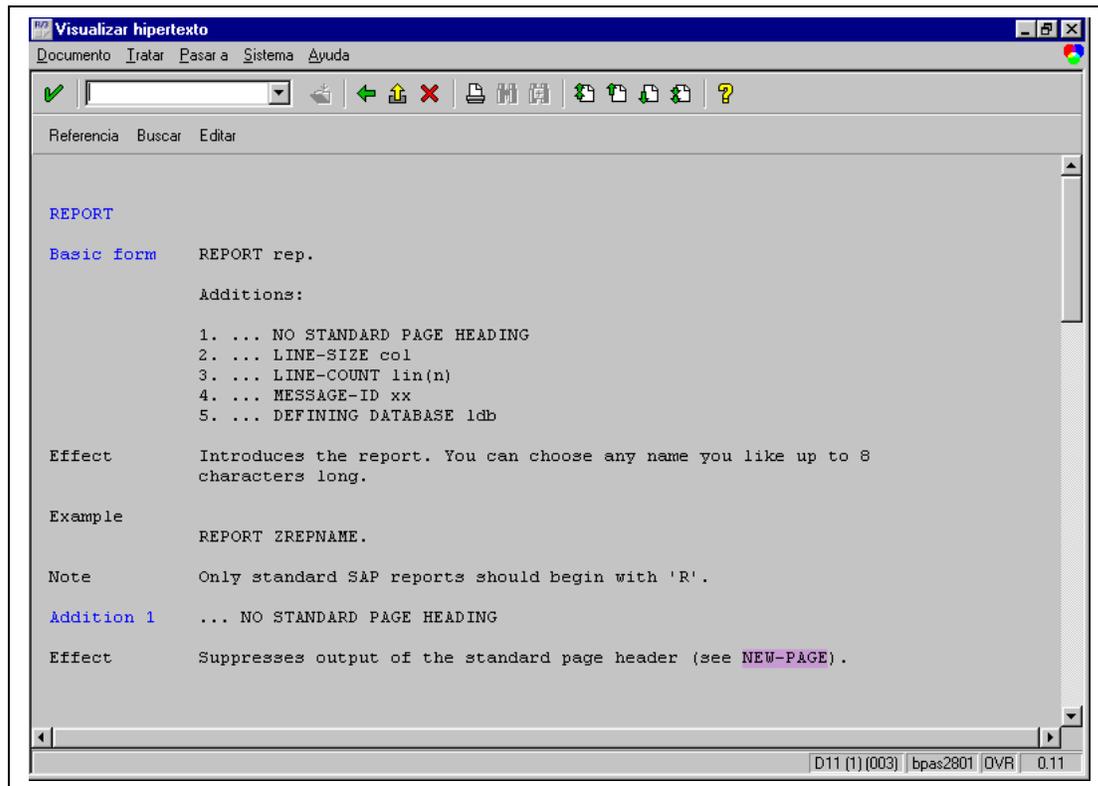
```
*****  
* PROGRAMA: ZREPORXX  
* DESCRIPCION: Muestra un listado de los clientes existentes en el  
*           que se detalla en nº de cliente junto con el nombre y  
*           apellidos.  
* AUTOR: Nombre y Apellidos           FECHA: 13/08/2001  
*-----*  
* CONTROL DE MODIFICACIONES  
* FECHA.       AUTOR.       DESCRIPCION MODIFICACION.  
*****
```

antes de la instrucción REPORT)

4.2.2 Definición de atributos de salida.

Todos los programas de este tipo empiezan con la instrucción 'REPORT' de ahí su nombre. Esta instrucción además de indicar el nombre e inicio del programa, permite señalar algunos de los atributos del programa como el alto y el ancho de listado, la clase de mensajes que utiliza etc...

Nota: Para obtener ayuda sobre una instrucción, situando el cursor sobre ella pulsamos F1 nos saldrá una pantalla de ayuda con la sintaxis y opciones de cada una de ellas. Para la sentencia REPORT se muestra la ayuda:



En nuestro ejemplo añadiremos a la instrucción `REPORT` las siguientes opciones:
`REPORT zreporxx NO STANDARD PAGE HEADING LINE-SIZE 120 LINE-COUNT 80.`

4.2.3 Definición de tablas externas.

Para utilizar tablas del diccionario de datos, es necesario hacer referencia a que tablas del diccionario se va a acceder desde el programa. Para ello será necesario indicar la palabra clave `TABLES` seguida del nombre o nombres de las tablas empleadas.

En nuestro ejemplo pondremos:

`TABLES: zclienxx.`

4.2.4 Pantalla de selección.

La 'pantalla de selección' es una pantalla en la que se indican los parámetros del programa. Estos parámetros van a utilizarse para 'filtrar' la información y procesar solamente aquella que pase el dicho 'filtro', es decir, cumpla con los criterios de selección.

En la pantalla de selección, podemos tener los siguientes elementos:

- *Parameters*: Recogen único valor, es decir, como criterio solamente se podrá indicar un único valor. Para incluir parámetros de selección podremos la palabra clave PARAMETERS seguida del nombre de parámetro y su tipo.

En nuestro ejemplo, vamos a definir un parámetro para seleccionar por Sociedad para ello escribiremos:

```
PARAMETERS p_bukrs LIKE zclienxx-bukrs OBLIGATORY DEFAULT '0001'.
```

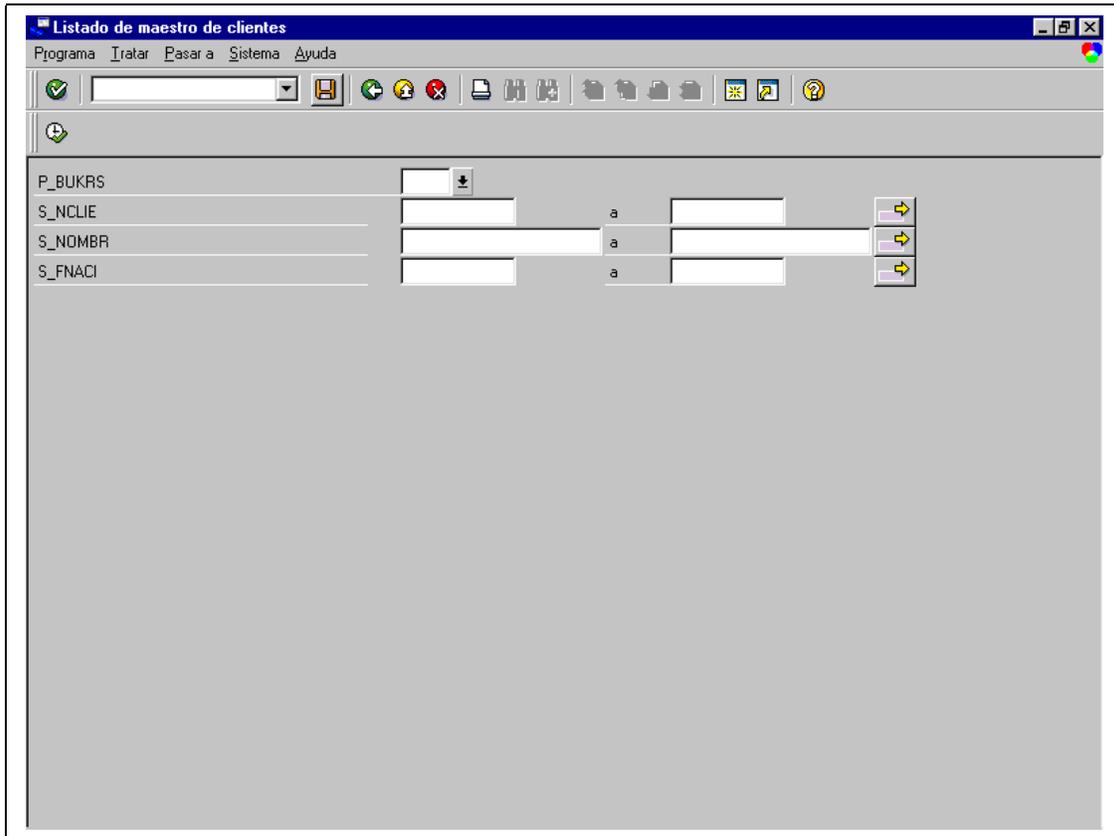
- *Select-options*: Recogen múltiples valores o un rango de valores, es decir, se pueden indicar más de un valor como criterio de selección. Se definen poniendo la palabra clave SELECT-OPTIONS seguida del nombre del rango de selección y el campo al que hace referencia.

Vamos a definir tres rangos de selección uno para el nº de cliente, otro para el nombre y otro para la fecha de nacimiento, para ello escribiremos:

SELECT-OPTIONS:

```
s_nclie FOR zclienxx-nclie,    " N° de cliente  
s_nombr FOR zclienxx-nombr,    " Nombre cliente  
s_fnaci FOR zclienxx-fnaci.    " Fecha de nacimiento
```

Si en este punto grabamos, verificamos, generamos y ejecutamos nuestro programa, únicamente tendríamos la pantalla de selección. Que en nuestro caso tendrá el siguiente aspecto:



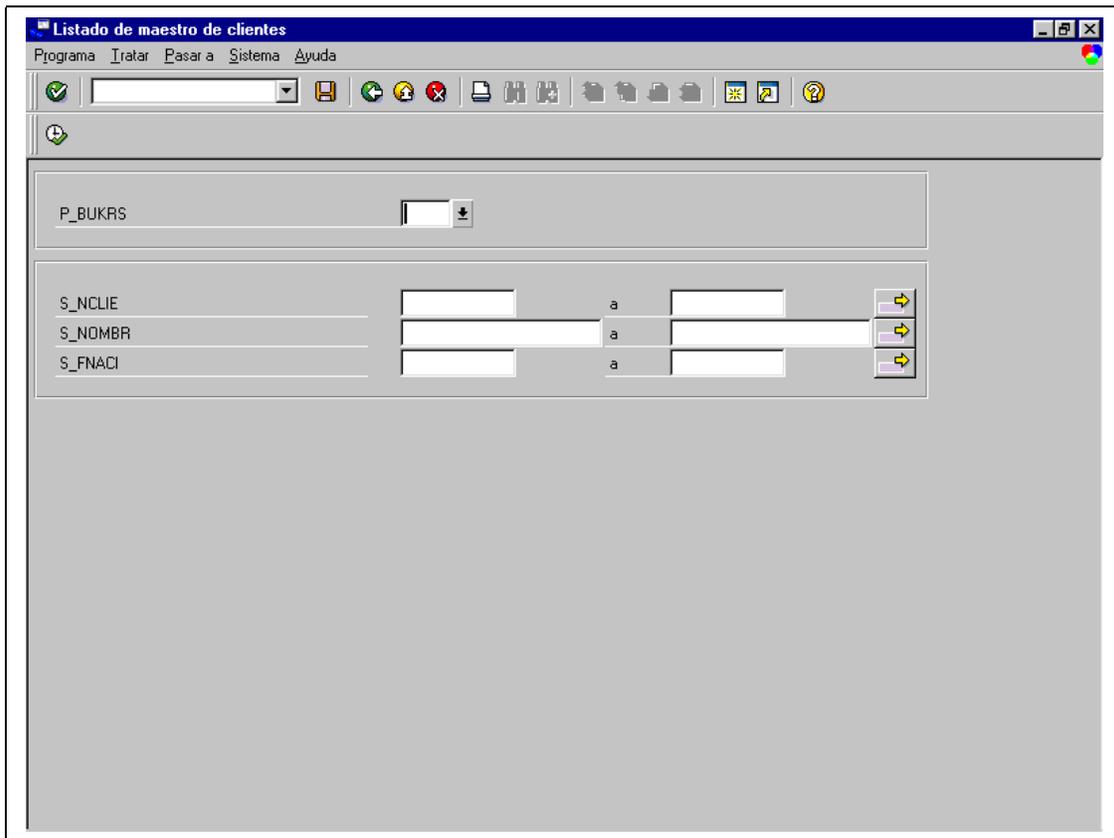
Podemos hacer que la pantalla de selección tenga un aspecto más elaborado para ello tenemos la posibilidad de incluir marcos, saltos de línea... Para conseguir estos efectos utilizaremos la palabra clave SELECTION SCREEN.

En nuestro ejemplo vamos a crear dos marcos uno que va a incluir al parámetro P_BUKRS y otro que va a contener a los rangos de selección S_NCLIEN, S_NOMBR y S_FNACI. Para ello incluiremos las siguientes instrucciones, de tal forma que tendremos:

```
SELECTION-SCREEN BEGIN OF BLOCK bloq1 WITH FRAME TITLE text-001.  
  PARAMETERS p_bukrs LIKE zclienxx-bukrs OBLIGATORY DEFAULT '0001'.  
SELECTION-SCREEN END OF BLOCK bloq1.
```

```
SELECTION-SCREEN BEGIN OF BLOCK bloq2 WITH FRAME TITLE text-002.  
  SELECT-OPTIONS:  
    s_nclie FOR zclienxx-nclie,      " N° de cliente  
    s_nombr FOR zclienxx-nombr,     " Nombre cliente  
    s_fnaci FOR zclienxx-fnaci.     " Fecha de nacimiento  
SELECTION-SCREEN END OF BLOCK bloq2.
```

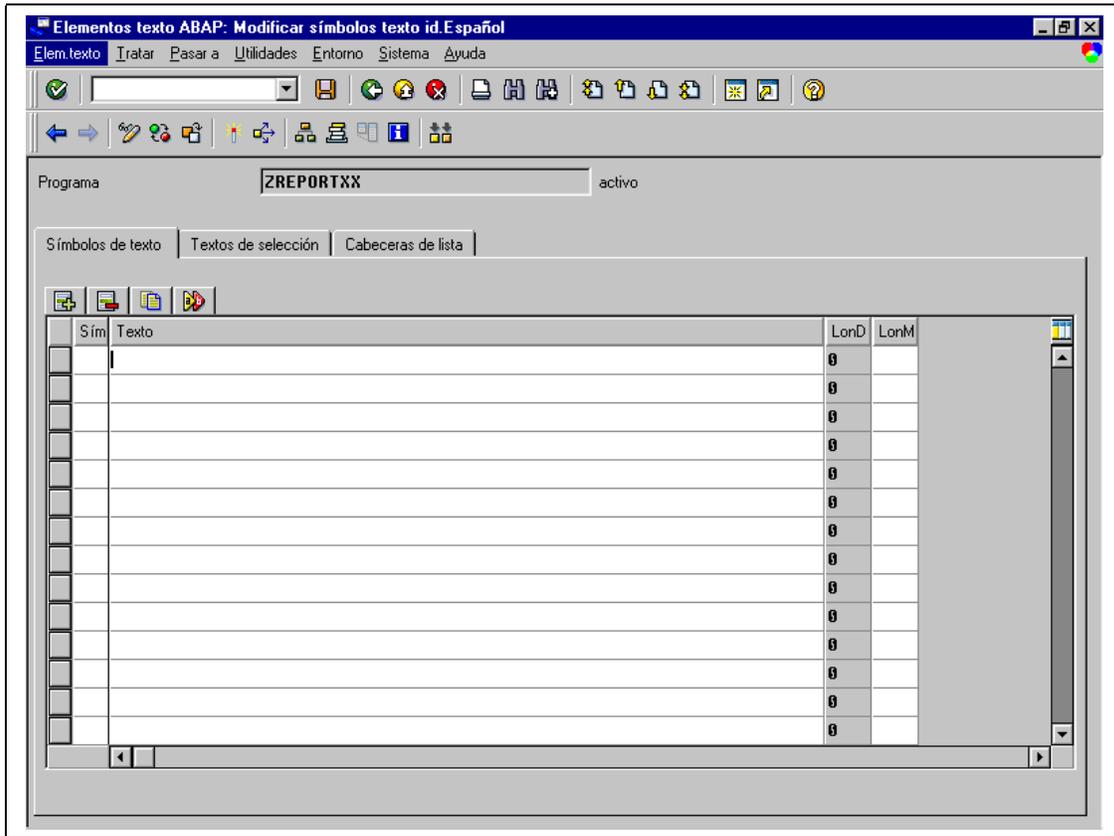
Conseguiremos que la pantalla tenga el siguiente aspecto:



Los marcos creados, no tienen texto esto es debido a que los símbolos de texto '001' y '002' referenciados con la palabra clave TEXT, no están definidos.

4.2.5 Elementos de texto

Pasar a →Elementos de texto→Símbolos de texto' (también se puede acceder desde la pantalla de acceso al editor seleccionando esta opción). Llegaremos a la siguiente pantalla:



4.2.5.1 Símbolos de texto.

Son cadenas de texto que se pueden utilizar a lo largo del programa para diferentes funciones. Para definir un símbolo de texto seleccionaremos la opción de menú la opción 'símbolos de texto' donde definiremos los símbolos de texto. El nº de símbolo puede estar compuesto por letras o números

Los campos LonD y LonM representan la longitud de la cadena introducida y la longitud máxima de salida del texto respectivamente, una vez introducido el texto, igualaremos el campo LonM con el valor de LonD.

En nuestro caso definiremos :

001 Datos sociedad

002 Datos cliente

Una vez introducidos los símbolos de texto grabamos  (F11).

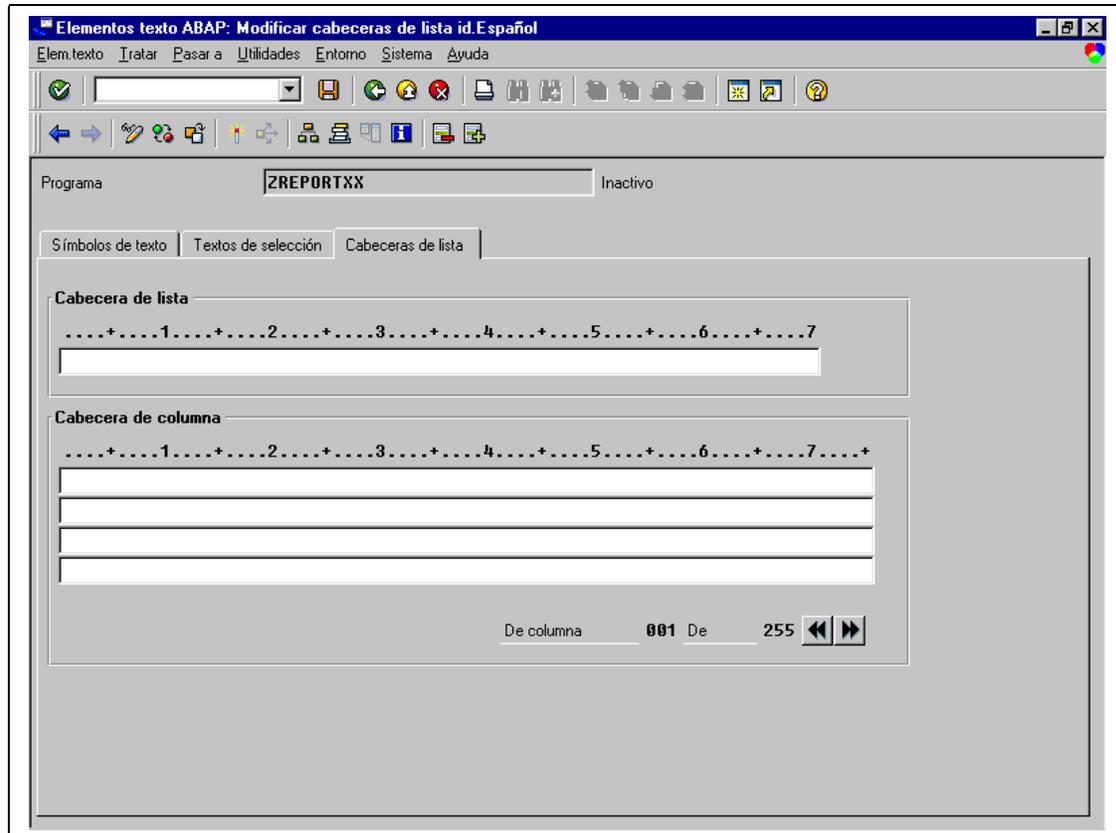
4.2.5.2 Textos de selección.

Son los textos que aparecerán en la pantalla de selección en lugar del nombre del parámetro. Si seleccionamos la pestaña 'Textos selección', aparecerán los parámetros y rangos de selección definidos en el programa.

En nuestro caso aparecerá la pantalla en la que completaremos los siguientes textos. Sociedad, Fecha de nacimiento, Número cliente, Nombre cliente. Una vez escritos los textos de selección grabamos  (F11).

4.2.5.3 Títulos y cabeceras.

Finalmente si seleccionamos la pestaña 'Cabeceras lista' tendremos



Cabecera de lista: Define el texto que aparecerá al comienzo de cada página.

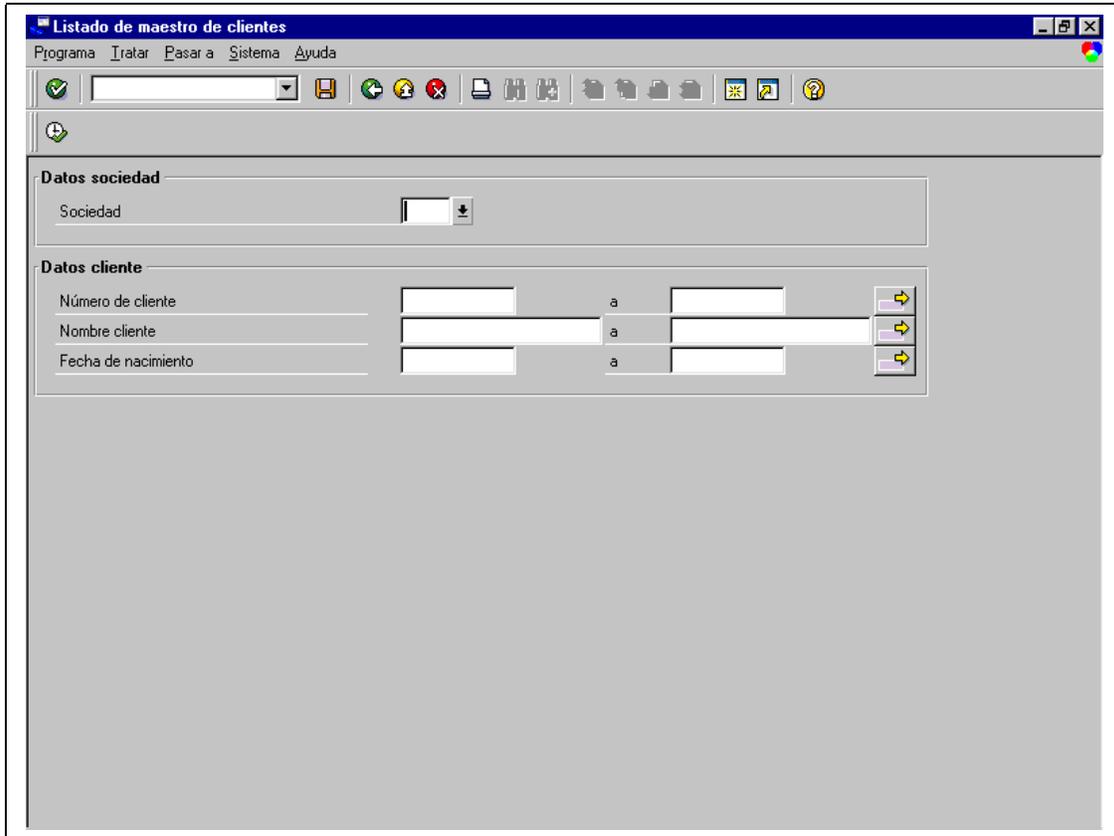
Pondremos 'Clientes seleccionados'.

Cabecera de columna: Define las líneas de cabecera de columna que aparecerán en cada una de las páginas, podemos definir hasta cuatro cabeceras de columna.

En nuestro ejemplo, pondremos 'C. Cliente Nombre Primer Apellido Segundo Apellido F.Nacimiento'. Reservando para cada campo la longitud de salida correspondiente. Una vez completados los títulos y cabeceras grabamos  (F11) y volvemos al editor del programa.

Las cabeceras de lista y de columna, no aparecerán en el listado, si se añade la opción NO STANDARD PAGE-HEADING en los atributos de salida del listado (Instrucción REPORT).

(Siguiendo con el ejemplo . Si ejecutamos a continuación el programa, veremos que la pantalla de selección a tomado el aspecto siguiente::)



4.2.6 Sentencias de salida de datos

Vamos a ver un resumen de las principales instrucciones para el formateo de datos por pantalla.

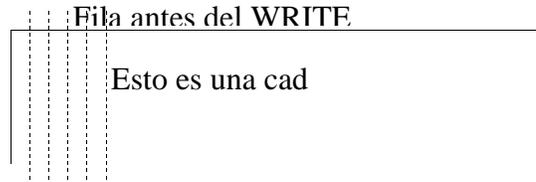
- **WRITE:**
Nos permite escribir el contenido de una variable declarada con DATA, el contenido de un registro de una tabla, el contenido de un FIELD-SYMBOL, un TEXT-SYMBOL o un literal en alguna posición del listado Las opciones a añadir a esta instrucción son:
 - ... AT pl (posición y longitud)
 - ... options (formato de salida)
 - ... ofmt (salida formateada para el campo)
 - ... AS CHECKBOX
 - ... AS SYMBOL
 - ... AS ICON
 - ... AS LINE
 - ... **AT /pl**

Con el símbolo (/) podemos hacer que ejecute un salto de línea. Con 'p' se especifica la columna de la línea actual en la que queremos que empiece a escribir y con 'l' la longitud máxima de lo que vamos a escribir.

Por ejemplo:

WRITE AT /5(15) 'Esto es una cadena'.

Resultado:



La posición y longitud se pueden expresar directamente con constantes numéricas (en este caso la palabra clave AT se puede omitir) y también con variables.

Si la longitud de salida es demasiado corta para el contenido de lo que se va a escribir la salida se rellenará de '*' si se trata de tipos P, I y F; en el resto de tipos se trucará el contenido por la derecha.

... options

Las opciones disponibles son:

- ... NO-ZERO
- ... NO-SIGN
- ... DD/MM/YY
- ... MM/DD/YY
- ... DD/MM/YYYY
- ... MM/DD/YYYY
- ... DDMMYY
- ... MMDDYY
- ... CURRENCY w
- ... DECIMALS d
- ... ROUND R
- ... UNIT u
- ... EXPONENT e
- ... USING EDIT MASK mask
- ... UNDER g
- ... NO-GAP
- ... LEFT-JUSTIFIED
- .. CENTERED
- .. RIGHT-JUSTIFIED

Por ejemplo para dar formato de salida a un importe con la moneda correspondiente:

WRITE D_IMPORTE CURRENCY D_MONEDA.

... *ofmt*

La salida se escribirá con los formatos de color, intensidad, etc. Las opciones disponibles en este punto se detallarán más adelante, dentro de la instrucción `FORMAT`.

Por ejemplo:

```
DATA F.  
WRITE F INVERSE COLOR 3.
```

Los formatos indicados únicamente aplican a la escritura de la instrucción actual.

- ... *AS CHECKBOX*

Hace que el campo escrito tenga la forma de es una caja de selección. La caja saldrá marcada o no marcada en función del valor del campo. (SPACE es no marcada y 'X' es marcada). Para poder hacer la caja de entrada/salida o solamente de salida añadiremos la opción `INPUT ON/OFF`.

Por ejemplo:

```
DATA D_CAJA(1) TYPE C VALUE SPACE.  
WRITE CAJA AS CHECKBOX. ó WRITE CAJA AS CHECKBOX INPUT  
OF.  
(Será de entrada/salida, valor por defecto)  
WRITE CAJA AS CHECKBOX INPUT OFF. (Será sólo de salida).
```

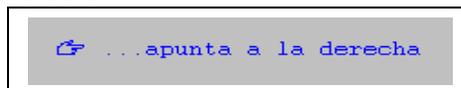
- ... *AS SYMBOL*

Permite escribir símbolos predefinidos. Para poder acceder a ellos será necesario incluir en nuestro programa la biblioteca de símbolos. Para ello incluiremos la sentencia `INCLUDE <SYMBOL>`. (para ver los símbolos disponibles buscaremos a través de la ayuda).

Por ejemplo, la sentencia:

```
WRITE: SYM_RIGHT_HAND AS SYMBOL, '...apunta a la derecha'.
```

Tendría como efecto:



- ... *AS ICON*

Similar a la opción anterior pero con la biblioteca de iconos. Par acceder a ella incluiremos la sentencia `INCLUDE <ICON>`. (Podemos ver los iconos con la ayuda).

Por ejemplo:

```
WRITE: ICON_GREEN_LIGHT AS ICON, '...Semáforo verde'.
```

Tendría como efecto:



- ... AS LINE

La forma de hacer líneas horizontales o verticales en un report y que éstas formen cuadrículas es escribir los caracteres SY-VLINE (línea vertical) y SY-ULINE (línea horizontal). Son equivalentes los caracteres '| 'y '-'. La forma exacta en la que aparecen estos segmentos depende de los caracteres adyacentes. Cuando tenemos un carácter de línea en una posición vertical y otro horizontal en la posición adyacente automáticamente se produce la unión de ambas. Si en la posición adyacente no hay otro carácter de línea ese carácter permanece inalterado.

En la mayoría de los casos esta técnica es suficiente para hacer cuadrículas, pero hay veces en las que las uniones no se producen de la forma en la que queremos. En esos casos será necesario echar mano de caracteres especiales como: LINE_TOP_LEFT_CORNER, LINE_BOTTOM_MIDDLE_CORNER que realizan esta operación. Para poder usar estos caracteres es necesario incluir INCLUDE <LINE>.

- **FORMAT.**

Sirve para definir los atributos de salida de lo que se va a escribir en la pantalla. A diferencia de la opción 'ofmt' del la instrucción WRITE el efecto de esta instrucción comprende todas las escrituras realizadas hasta la siguiente sentencia FORMAT. (Los cambios de evento, START-OF-SELECTION, TOP-OF-PAGE... provocan que se inicialicen todas las opciones). Las posibles opciones de la sentencia FORMAT son:

FORMAT INTENSIFIED ON/OFF	Esta opción modifica el color de fondo.
FORMAT INVERSE ON/OFF	Invierte el color de fondo y el texto escrito.
FORMAT INPUT ON/OFF	Habilita o deshabilita para entrada en los campos que se escriban a continuación.
FORMAT HOTSPOT ON/OFF	Habilita o deshabilita el formato del cursor cuando se sitúe sobre las líneas o palabras que se escriban a continuación tome aspecto de <i>mano</i>  .
FORMAT RESET	Inicializa todas las opciones de formato. Esta instrucción equivale a: FORMAT COLOR OFF INTENSIFIED OFF INVERSE OFF HOTSPOT OFF INPUT OFF.

Ejemplo:

```
FORMAT COLOR COL_BACKGROUND ON.
```

```
WRITE F. " Se escribirá en color BACK..
```

```
WRITE D. " Se escribirá en color BACK..
```

```
FORMAT COLOR COL_BACKGROUND OFF.
```

```
WRITE F. " Se escribirá en color normal
```

- **ULINE**

Escribe una línea horizontal en la línea actual. Por defecto el tamaño de la línea comprende todo el ancho del listado. Podemos seleccionar el ámbito de la línea con añadiendo AT pl donde:

p	Se corresponde con la columna en la que comienza la línea.
l	Se corresponde con la longitud que tiene la línea.

Las siguientes instrucciones producen el mismo resultado:

```
-ULINE AT 3(10).
```

```
-WRITE AT 3(10) SY-ULINE.
```

- SKIP y POSITION

Estas dos instrucciones SKIP y POSITION. Permiten situar el puntero de escritura en una posición determinada.

POSITION n Hace que el puntero se sitúe en la columna n.

SKIP n Hace que el puntero salte n líneas.

SKIP TO LINE n Hace que el puntero se mueva a la línea n del listado.

Las siguientes instrucciones producen un resultado equivalente a la instrucción (WRITE AT /5(15) 'Esto es una cadena'.)

```
DATA CADENA TYPE C VALUE 'Esto es una cadena'.
```

```
SKIP.
```

```
POSITION 5.
```

```
WRITE (15) CADENA.
```

- NEW-LINE

Provoca una nueva línea en el listado.

Mediante la opción NO-SCROLLING podemos hacer que la línea permanezca fija durante el desplazamiento (Scroll) horizontal. (El valor por defecto es que se pueda desplazar por tanto la opción SCROLLING se suele omitir).

- NEW-PAGE

Provoca una nueva página en el listado. Tiene varias opciones mediante las cuales se puede determinar el tamaño (nº filas y nº de columnas) de la nueva página, si se imprime o no el título....

- **RESERVE**

Sirve para reservar un cierto nº de líneas antes de un salto de página. RESERVE n LINES. Si en la página actual no hay n líneas disponibles entonces se provocará un salto de página. De esta forma se puede asegurar que las n líneas escritas a continuación estarán en la misma página.

- **MESSAGE.**

Sirve para mostrar un determinado mensaje de una clase (colección) de mensajes. El mensaje puede ser mostrado en la barra de status, o en forma de ventana, dependiendo del tipo de mensaje. Existen los siguientes tipos de mensajes:

- I Informativos.
- W De precaución
- E Error
- A Terminación
- X De salida
- S Éxito.

Con un mensaje de tipo 'I', se mostrará una ventana en la que visualiza el contenido del mensaje y continúa con la ejecución del programa, con 'S' se muestra el mensaje en la barra de status y continua el programa, con un tipo 'E' se muestra un mensaje y puede terminar la ejecución del programa, depende del evento en el que este situado, un tipo 'W' en un listado tiene el mismo significado, un mensaje tipo 'A' muestra el mensaje y termina el programa, finalmente un tipo 'X' muestra aborta el programa provocando un 'DUMB'.

Para indicar un tipo u otro podemos usar la sintaxis:
MESSAGE TNNN(XX) with p1 p2 p3 p4.

Donde T representa el tipo de mensaje (I, S, E...).
NNN representa el nº de mensaje.
XX representa la clase de mensajes. (Podemos no indicar esta clase de mensajes si se añade la opción MESSAGE-ID de la instrucción REPORT).
p1..p4 representan los parámetros del mensaje.

La definición del mensaje dentro de la clase consiste en un texto indicativo del mensaje junto a unos parámetros si los tiene:

Por ejemplo, podríamos definir un mensaje con el número 001 de la clase ZZ 'El cliente '& 'No existe.

Llamaríamos al mensaje con MESSAGE I001(ZZ) WITH D_NCLIENTE.
Para hacerlo informativo y MESSAGE E001(ZZ) WITH D_NCLIENTE.
Para hacerlo de tipo error.

4.2.7 Eventos

4.2.7.1 INITIALIZATION

Se ejecuta antes de mostrarse la pantalla de selección. Normalmente se utiliza para inicializar las variables de programa y las opciones de la pantalla de selección.

4.2.7.2 START-OF-SELECTION

Este evento se lanza cuando se procesa la instrucción *REPORT...* En ese momento se empieza a ejecutar el código que se encuentra entre *REPORT* y la palabra *START-OF-SELECTION*. Inmediatamente después se procesa el bloque contenido dentro de este evento.

En este evento se sitúan las instrucciones de acceso a bases de datos una vez terminada este código se ejecuta el evento END-OF-SELECTION (Si está definido).

(Siguiendo con nuestro ejemplo, vamos a realizar la selección de los clientes que cumple los criterios definidos por los parámetros de selección. Los datos de estos clientes los vamos a almacenar en una tabla interna.

Definimos la siguiente tabla interna que situaremos en la parte correspondiente del programa.

* Tabla interna de datos a listar

```
DATA: BEGIN OF I_ZCLIENXX OCCURS 0.  
      INCLUDE STRUCTURE ZCLIENXX.  
DATA: END OF I_ZCLIENXX.
```

Añadimos el evento START-OF-SELECTION en la parte del programa que le corresponde con el siguiente código:
START-OF-SELECTION.

* Inicializamos la tabla interna.

```
REFRESH I_ZCLIENXX.  
CLEAR I_ZCLIENXX.
```

* Seleccionamos los datos de los clientes que cumplen criterios

```
SELECT *  
FROM ZCLIENXX INTO TABLE I_ZCLIENXX  
WHERE BUKRS = P_BUKRS  
AND NCLIE IN S_NCLIE  
AND NOMBR IN S_NOMBR  
AND FNACI IN S_FNACI.
```

* Comprobamos si no hay datos seleccionados en cuyo caso,

* escribiremos un texto que lo indique.

```
IF NOT ( SY-SUBRC = 0 ).  
  WRITE / TEXT-003. " No hay datos para la selección indicada  
ENDIF.
```

4.2.7.3 END-OF-SELECTION.

El código asociado a este evento se procesa cuando se termina la selección de datos de tablas o bases de datos lógicas, es decir, cuando termina el evento START-OF-SELECTION.

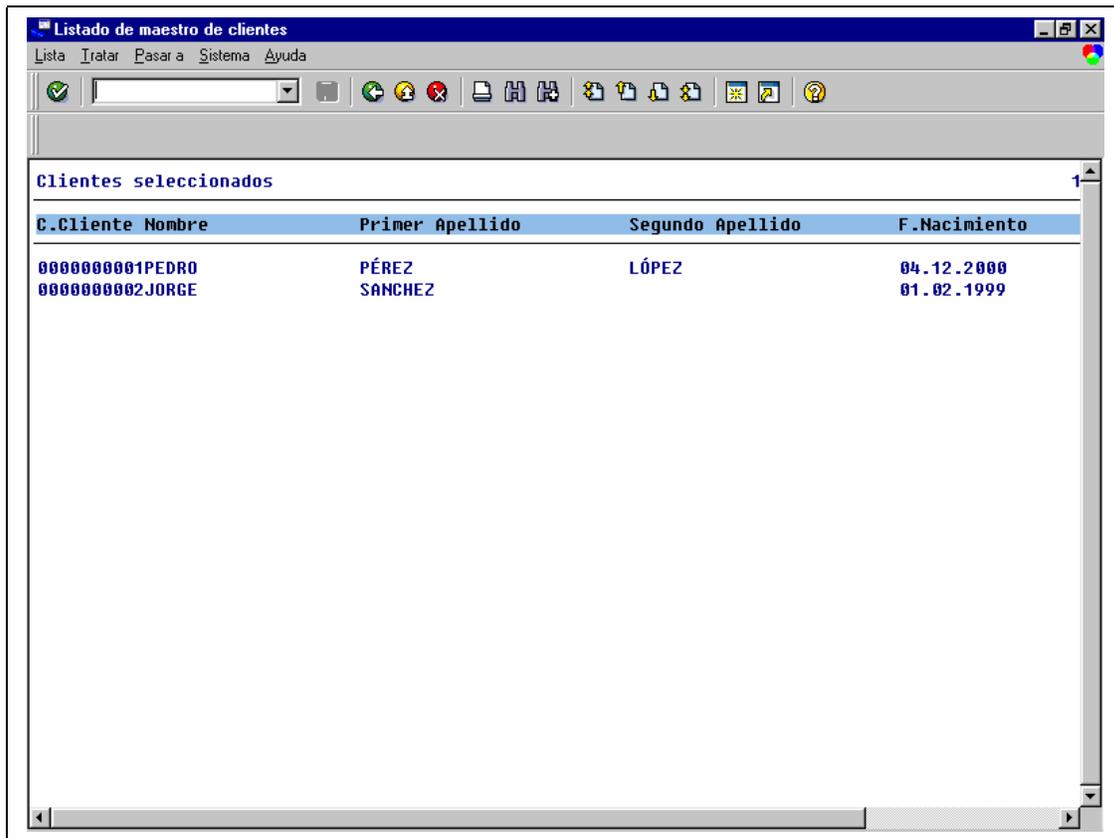
(En nuestro ejemplo escribiremos los datos de los clientes seleccionados, para ello añadiremos el siguiente código:
END-OF-SELECTION.

* Vamos a escribir el listado con los clientes seleccionados en la

* Tabla interna I_ZCLIENXX

```
LOOP AT I_ZCLIENXX.  
  WRITE: /1(10) I_ZCLIENXX-NCLIE,  
         11(20) I_ZCLIENXX-NOMBR,  
         31(25) I_ZCLIENXX-APEL1,  
         56(25) I_ZCLIENXX-APEL2,  
         81(10) I_ZCLIENXX-FNACI.  
ENDLOOP.
```

Si llegado este punto, ejecutamos el programa, si hay datos en la tabla maestro de clientes y cumplen los criterios de selección aparecerá un listado algo así:



C.Cliente	Nombre	Primer Apellido	Segundo Apellido	F.Nacimiento
000000001	PEDRO	PÉREZ	LÓPEZ	04.12.2000
000000002	JORGE	SANCHEZ		01.02.1999

4.2.7.4 TOP-OF-PAGE

Se ejecuta al inicio de cada nueva página.

Se suele utilizar para dar formato a la cabecera de página, cuando no se utilizan las definidas en 'Títulos y cabeceras'. (Se añade la opción NO STANDARD PAGE HEADING).

Se le puede añadir la opción DURING LINE-SELECTION, este evento se producirá al inicio de una nueva página de un nivel de lista secundaria. (Reports interactivos).

Se pueden tener por tanto dos eventos,
TOP-OF-PAGE que actúa en el nivel inicial.

TOP-OF-PAGE DURING LINE-SELECTION que actúa en los de niveles superiores.

4.2.7.5 END-OF-PAGE

Se ejecuta cuando una página finaliza bien porque no tiene más líneas bien porque se ejecuta la instrucción RESERVE n LINES. Este evento no se ejecutará si el nº de líneas de la página no está definido en la sentencia REPORT, tampoco se ejecutará se fuerza una nueva página con NEW-PAGE.

4.2.7.6 AT SELECTION-SCREEN

Se utiliza para controlar la pantalla de selección.
Sus opciones permiten mostrar valores de ayuda en los campos de selección, mostrar/ocultar campos de selección....

(En nuestro ejemplo, vamos a dar la posibilidad de mostrar los valores posibles para el rango de selección nombre del cliente. De esta forma aparecerá, al situarnos sobre el campo, el botón de ayuda de valores posibles  (F4), al pulsarlo mostrará una ventana con los distintos nombres existentes en nuestra tabla.

Para ello añadiremos el siguiente código:

```
* Valores posibles para el rango S_Nombre valor desde.  
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-LOW.  
* Llamada a la rutina que muestra los valores existentes  
  PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-LOW.  
  
* Valores posibles para el rango S_Nombre valor hasta  
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-HIGH.  
* Llamada a la rutina que muestra los valores existentes  
  PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-HIGH.
```

Los rangos de selección tiene implícitos dos valores el mínimo y el máximo, para que aparezca la ayuda en los dos es necesario añadir dos eventos, uno para cada valor. La funcionalidad en ambos casos es la misma por tanto es susceptible de se introducida en un único procedimiento.

(En nuestro caso lo llamamos VALORES_POSIBLES_NOMBRE, ver el código correspondiente en el listado completo del programa).

Para implementar este procedimiento, nos hemos ayudado con una función predefinida en el STANDARD. 'HELP_VALUES_GET_WITH_TABLE'. Esta función permite mostrar datos en una ventana y seleccionar uno de ellos. Permite mostrar información de más de un campo, para ello en la tabla interna de definición de campos añadiremos tantas entradas como campos deseemos visualizar. Para indicar los valores lo haremos de forma secuencial es decir, en la entrada uno, el valor para el campo 1 en la entrada 2 el valor para el campo 2 ... así hasta completar todos los campos, a continuación otra vez la entrada para el campo 1, campo 2 y así sucesivamente para cada secuencia .

4.2.8 Ejemplo de listado.

Vamos a dar otra apariencia a nuestro listado, para ello vamos a crear una cabecera propia y no vamos a utilizar la definida en 'Título y Cabeceras'. En primer lugar para desactivar esta cabecera, añadiremos la opción NO STANDARD PAGE HEADING a la instrucción REPORT. Por otra parte vamos a hacer que el campo correspondiente al nº de cliente permanezca fijo durante el desplazamiento horizontal por el informe. En la

pantalla de selección, vamos a utilizar el match code de búsqueda de clientes creado 'ZCXX'.

El código resultante será algo así:

```
*****
* PROGRAMA: ZREPORXX.
* DESCRIPCION: Muestra un listado de los clientes existentes en el
* que se detalla en nº de cliente junto con el nombre y
* apellidos
* AUTOR : MAD00 FECHA: 13/08/2001
*-----*
* CONTROL DE MODIFICACIONES
* FECHA. AUTOR. DESCRIPCION MODIFICACION.
*****
REPORT zreporxx NO STANDARD PAGE HEADING LINE-SIZE 120 LINE-COUNT 80.

*****
* Tablas del diccionario de datos
*****
TABLES: zclienx. " Maestro de clientes.

*****
* Definición de constantes
*****
CONSTANTS:
  c_pos_nclie(3) TYPE n VALUE 1, " Posición nº cliente
  c_pos_nombr(3) TYPE n VALUE 12, " Posición nombre
  c_pos_apel1(3) TYPE n VALUE 33, " Posición apellido1
  c_pos_apel2(3) TYPE n VALUE 59, " Posición apellido2
  c_pos_fnaci(3) TYPE n VALUE 85, " Posición F. Nacimiento
  c_pos_final(3) TYPE n VALUE 96, " Posición final
  c_ancho_total(3) TYPE n VALUE 96, " Ancho total del informe,
  c_pos_titul(3) TYPE n VALUE 1, " Posición título,
  c_pos_usuar(3) TYPE n VALUE 37, " Posición usuario
  c_pos_fecha(3) TYPE n VALUE 60, " Posición fecha
  c_pos_pago(3) TYPE n VALUE 85. " Posición nº de página.

*****
* Tablas internas
*****

* Tabla interna de datos a listar
DATA: BEGIN OF i_zclienx OCCURS 0.
      INCLUDE STRUCTURE zclienx.
DATA: END OF i_zclienx.

*****
```

* Pantalla de selección *

* Parámetros

SELECTION-SCREEN BEGIN OF BLOCK bloq1 WITH FRAME TITLE text-001.

* Sociedad

PARAMETERS p_bukrs LIKE zclienxx-bukrs OBLIGATORY DEFAULT '0001'.

SELECTION-SCREEN END OF BLOCK bloq1.

* Rangos de selección

SELECTION-SCREEN BEGIN OF BLOCK bloq2 WITH FRAME TITLE text-002.

SELECT-OPTIONS:

* Añadimos el match-code para el código de cliente

s_nclie FOR zclienxx-nclie MATCHCODE OBJECT zcxx, " N° de cliente

s_nombr FOR zclienxx-nombr, " Nombre cliente

s_fnaci FOR zclienxx-fnaci. " Fecha de nacimiento

SELECTION-SCREEN END OF BLOCK bloq2.

* Comienzo de selección *

START-OF-SELECTION.

* Inicializamos la tabla interna.

REFRESH i_zclienxx.

CLEAR i_zclienxx.

* Seleccionamos los datos de los clientes que cumplen criterios

SELECT *

FROM zclienxx INTO TABLE i_zclienxx

WHERE bukrs = p_bukrs

AND nclie IN s_nclie

AND nombr IN s_nombr

AND fnaci IN s_fnaci.

* Comprobamos si no hay datos seleccionados en cuyo caso,

* escribiremos un texto que lo indique.

IF NOT (sy-subrc = 0).

WRITE / text-003. " No hay datos para la selección indicada

ENDIF.

```
* Final de la selección *
*****
END-OF-SELECTION.

* Vamos a escribir el listado con los clientes seleccionados en la
* tabla interna I_ZCLIENXX
LOOP AT i_zclienxx.

* Escribimos el nº de cliente en color clave, el resto en otro color
FORMAT COLOR COL_KEY.
WRITE: AT /c_pos_nclie '|' NO-GAP, i_zclienxx-nclie.
FORMAT COLOR COL_NORMAL.

WRITE: AT c_pos_nombr '|' NO-GAP, i_zclienxx-nombr,
      AT c_pos_apel1 '|' NO-GAP, i_zclienxx-apel1,
      AT c_pos_apel2 '|' NO-GAP, i_zclienxx-apel2,
      AT c_pos_fnaci '|' NO-GAP, i_zclienxx-fnaci,
      AT c_pos_final '|' NO-GAP.
ENDLOOP.

* Si se han seleccionado datos, escribimos la línea final.
IF ( sy-subrc = 0 ).
  ULINE AT /c_pos_nclie(c_ancho_total).
ENDIF.

*****
* Cabecera de página *
*****
TOP-OF-PAGE.

* Hacemos que la línea permanezca fija en el scroll horizontal
NEW-LINE NO-SCROLLING.

* Activamos el color de cabecera
FORMAT COLOR COL_HEADING INTENSIFIED OFF.

* Escribimos información de cabecera resaltando los valores
* Título y sociedad
WRITE: AT c_pos_titul text-009.
WRITE p_bukrs INTENSIFIED ON.

* Usuario de creación del informe
WRITE AT c_pos_usuar text-010.
WRITE sy-uname INTENSIFIED ON.

* Fecha de creación
WRITE AT c_pos_fecha text-011.
WRITE sy-datum INTENSIFIED ON USING EDIT MASK '___/___/___'.
```

* **Nº de página.**

WRITE AT c_pos_pagno text-012.
WRITE sy-pagno INTENSIFIED ON.

* **Fecha de creación**

WRITE AT c_pos_fecha text-011.
WRITE sy-datum INTENSIFIED ON.

* **Nº de página.**

WRITE AT c_pos_pagno text-012.
WRITE sy-pagno INTENSIFIED ON.

* **Escribimos una línea horizontal**

ULINE AT /c_pos_nclie(c_ancho_total).

* **Activamos el color de cabeceras intensificado**

FORMAT COLOR COL_HEADING INTENSIFIED ON.

* **Escribimos la cabecera de la columnas separadas por el carácter '|'**.

* **Hacemos que la columna de nº de cliente sea fija en el scroll horiz.**

WRITE: AT /c_pos_nclie '|' NO-GAP, text-004.

* **Hasta la columna hacemos fijo**

SET LEFT SCROLL-BOUNDARY.

* **Escribimos el detalle de cabecera de cliente**

WRITE: AT /c_pos_nclie '|' NO-GAP, text-004,
AT c_pos_nombr '|' NO-GAP, text-005,
AT c_pos_apel1 '|' NO-GAP, text-006,
AT c_pos_apel2 '|' NO-GAP, text-007,
AT c_pos_fnaci '|' NO-GAP, text-008,
AT c_pos_final '|' NO-GAP.

* **Escribimos una línea horizontal**

ULINE AT /c_pos_nclie(c_ancho_total).

* **Control de la pantalla de selección.**

* **Valores posibles para el rango S_Nombre valor desde.**

AT SELECTION-SCREEN ON VALUE-REQUEST FOR s_nombr-low.

* **Llamada a la rutina que muestra los valores existentes**

PERFORM valores_posibles_nombre USING s_nombr-low.

* **Valores posibles para el rango S_Nombre valor hasta**

AT SELECTION-SCREEN ON VALUE-REQUEST FOR s_nombr-high.

* Llamada a la rutina que muestra los valores existentes
 PERFORM valores_posibles_nombre USING s_nombr-high.

* Rutinas adicionales. *

&-----

*& Form VALORES_POSIBLES_NOMBRE

&-----

* Muestra una ventana con los distintos nombres existentes en *
 * la tabla maestro de clientes ZCLIENXX. Dando la posibilidad de *
 * seleccionar uno de ellos. Devuelve en el parámetro PS_NOMBR el *
 * nombre seleccionado o vacío si no se realiza selección. *

* -->PS_NOMBR Nombre seleccionado *

FORM valores_posibles_nombre USING ps_nombr LIKE zclienxx-nombr.

* Variables locales *

* Tabla interna para almacenar los campos y atributos a visualizar.

```
DATA : BEGIN OF i_fields OCCURS 1.
      INCLUDE STRUCTURE help_value.
DATA: END OF i_fields.
```

* Tabla interna para almacenar el valor de campos a mostrar.

```
DATA : BEGIN OF i_values OCCURS 0,
      valor(20) TYPE c,
      END OF i_values.
```

* Proceso *

* Insertamos en la tabla I_FIELDS los atributos del campo a mostrar

```
CLEAR i_fields.
i_fields-tabname = 'ZCLIENXX'. " Nombre de la tabla
i_fields-fieldname = 'NOMBR'. " Nombre del campo
i_fields-selectflag = 'X'. " Valor seleccionable
APPEND i_fields.
```

* Seleccionamos los distintos nombres existentes en la tabla ZCLIENXX

```
* y los almacenamos en la tabla de valores
SELECT DISTINCT nombr INTO zclienxx-nombr
FROM zclienxx
WHERE bukrs = p_bukrs.
i_values-valor = zclienxx-nombr.
```

```
APPEND i_values.  
ENDSELECT.
```

* Llamamos a la función predefinida que muestra los datos

```
CALL FUNCTION 'HELP_VALUES_GET_WITH_TABLE'
```

```
* EXPORTING  
*   CUCOL           = 0  
*   CUROW           = 0  
*   DISPLAY         = ''  
*   FIELDNAME       = ''  
*   TABNAME         = ''  
*   NO_MARKING_OF_CHECKVALUE = ''  
*   TITLE_IN_VALUES_LIST = ''  
*   TITEL           = ''  
*   SHOW_ALL_VALUES_AT_FIRST_TIME = ''  
*   NO_CONVERSION   = ''
```

```
IMPORTING
```

```
  select_value      = ps_nombr
```

```
TABLES
```

```
  fields            = i_fields
```

```
  valuetab         = i_values
```

```
EXCEPTIONS
```

```
  field_not_in_ddic      = 1
```

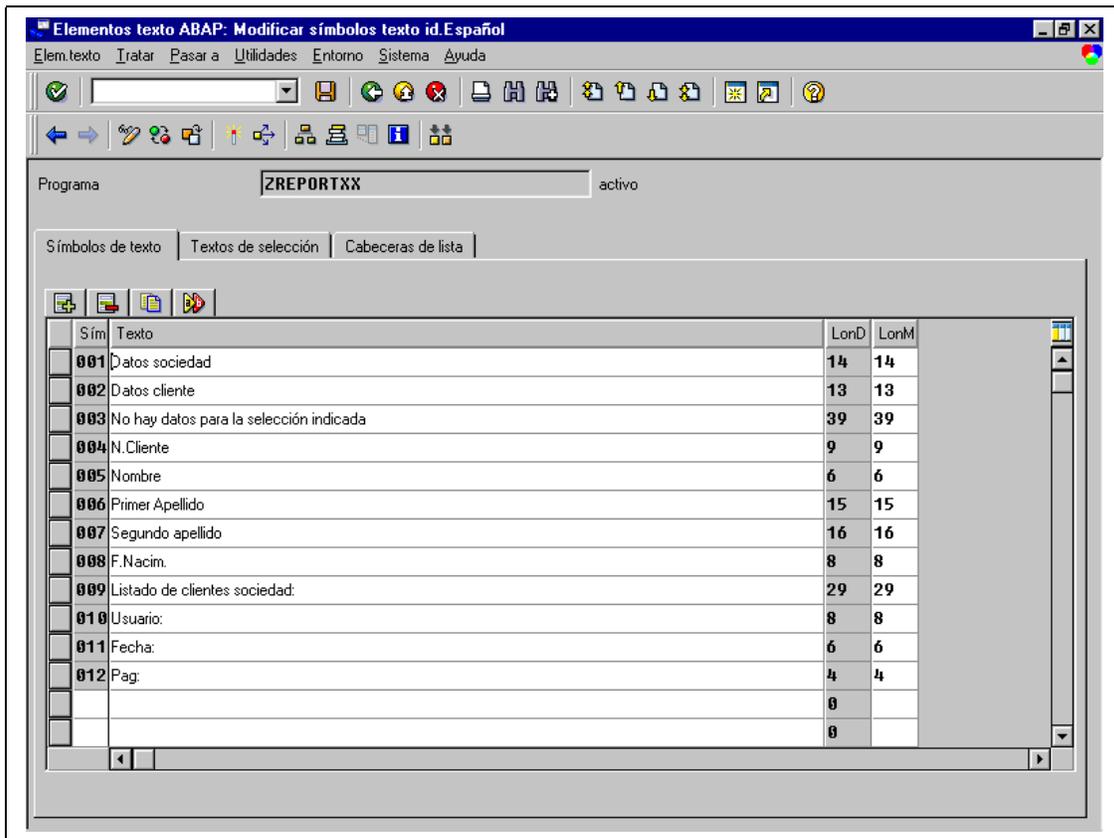
```
  more_than_one_selectfield = 2
```

```
  no_selectfield        = 3
```

```
  OTHERS                = 4.
```

```
ENDFORM.          " VALORES_POSIBLES_NOMBRE
```

Será necesario definir los símbolos de texto:



El listado tendrá ahora el siguiente aspecto:

N.Cliente	Nombre	Primer Apellido	Segundo apellido	F.Nacim.
000000001	PEDRO	PÉREZ	LÓPEZ	04.12.2000
000000002	JORGE	SANCHEZ		01.02.1999

4.2.9 Eventos de rupturas de secuencia en tablas internas.

Para el tratamiento de los registros de una tabla interna con la instrucción LOOP-ENDLOOP, podemos utilizar las siguientes instrucciones de control. Para que el funcionamiento de los eventos AT NEW y AT END sea correcto, la tabla interna ha de estar ordenada por los campos controlados.

- **AT FIRST ... ENDAT.**
Este evento se ejecuta para el primer registro de la tabla interna, por tanto las instrucciones escritas en este bloque únicamente se ejecutarán una vez. (Dentro del evento, no estará disponible el valor de ningún campo de la tabla interna).
- **AT LAST ... ENDAT**
Este evento se ejecuta cuando se está procesando el último registro de la tabla interna, por tanto también sus instrucciones se ejecutarán una única vez. (Dentro del evento, no estará disponible el valor de ningún campo de la tabla interna).
- **AT NEW campo ... ENDAT.**
Este evento se ejecuta si el contenido de la cabecera de la tabla interna desde el primer campo hasta el campo 'campo' incluido es distinta a la del paso anterior, es decir, si cambia la 'subcabecera' formada por estos campos. (Dentro

del evento, estarán disponibles los valores de los campos de esta 'subcabecera', el resto de campos, no estarán disponibles).

- **AT END OF campo ... ENDAT.**

Tiene el mismo funcionamiento que AT NEW salvo que se ejecuta cuando se detecta que el siguiente registro no coincide con el registro actual. (Estarán disponibles los valores de los campos formados por la 'subcabecera').

Dentro de estos eventos podemos utilizar la instrucción SUM, que lo que hace es sumar todos los campo numéricos de la tabla interna, es decir, acumula en el campo actual de la tabla la suma de los registros tratados hasta la entrada en el evento.

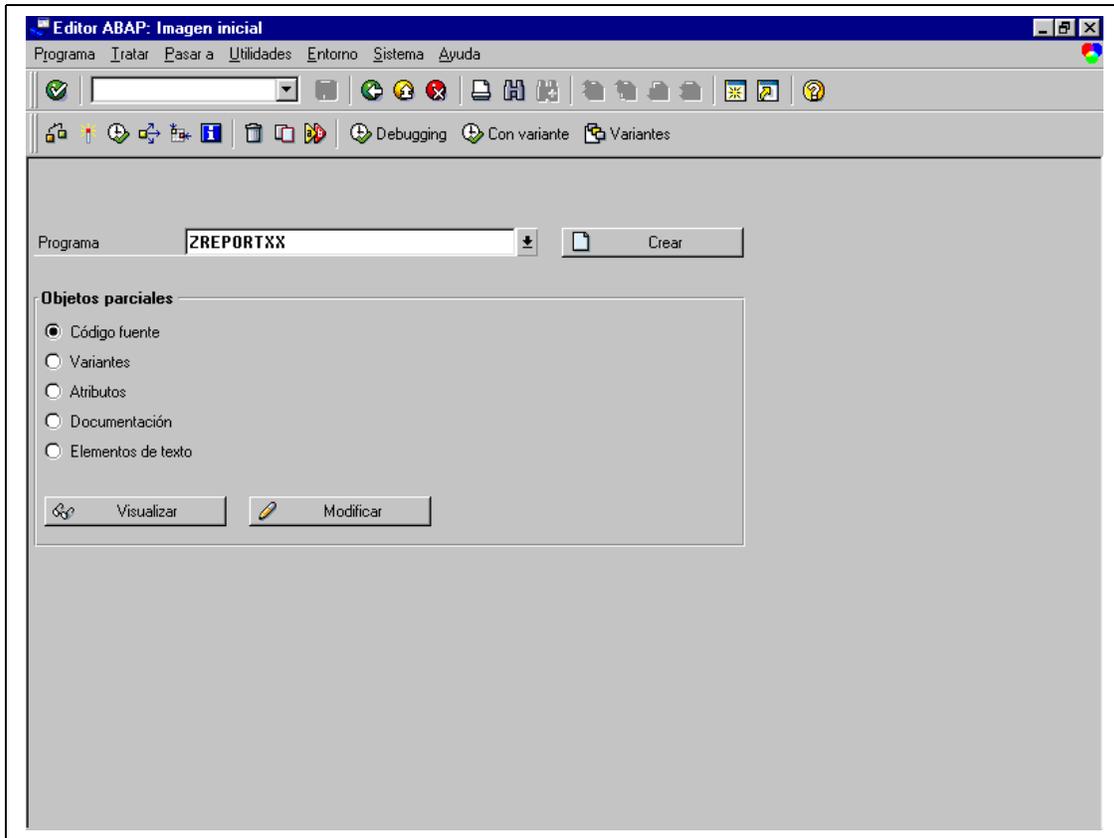
- **ON CHANGE OF campo ... ENDON.**

Este evento se ejecuta cada vez que el valor del campo no coincide con el valor anterior. (Este evento no exclusivo de las tablas internas, también se puede utilizar en acceso a base datos lógicas, en SELECT... ENDSELECT,...).

Vamos a ver un ejemplo de utilización de estos eventos. Vamos a mostrar las facturas de cada uno de los clientes, totalizando por moneda.

Para ello vamos a partir del programa anterior y haremos una copia, el nuevo programa se llamará 'ZREPO1XX'.

Para hacer una copia de un programa, en la pantalla de acceso al editor de programas:



Pulsamos el botón copiar  (Ctrl+F5), aparecerá una ventana en la que escribiremos el programa fuente y el programa destino. A continuación aparecerá otra ventana para indicar que elementos se desean copiar, dejamos las opciones por defecto y pulsamos la opción de copiar (como objeto local).

Sobre el programa copia 'ZREPO1XX' haremos las siguientes modificaciones:

Añadiremos la tabla de facturas en la sentencia TABLES, crearemos una tabla interna para almacenar los datos de las facturas, modificaremos el evento END_OF_SELECTION para recuperar e imprimir las facturas y TOP-OF-PAGE para escribir el detalle de segunda línea de cabecera.

Es decir, el programa modificado será algo así:

```
*****
* PROGRAMA: ZREPO1 XX. *
* DESCRIPCION: Muestra un listado de los clientes existentes en el *
* que se detalla en nº de cliente junto con el nombre y *
* apellidos, además, muestra el detalle de cada una de *
```

* las facturas de los clientes totalizadas por moneda fac. *
* AUTOR : MAD00 FECHA: 13/08/2001 *

* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *

REPORT zrepo1xx NO STANDARD PAGE HEADING LINE-SIZE 120 LINE-COUNT 80.

* Tablas del diccionario de datos *

TABLES: zclienxx, " Maestro de clientes.
 zfactuxx. " Facturas

* Definición de constantes *

CONSTANTS:

- c_pos_nclie(3) TYPE n VALUE 1, " Posición nº cliente
- c_pos_nombr(3) TYPE n VALUE 12, " Posición nombre
- c_pos_apel1(3) TYPE n VALUE 33, " Posición apellido1
- c_pos_apel2(3) TYPE n VALUE 59, " Posición apellido2
- c_pos_fnaci(3) TYPE n VALUE 85, " Posición F. Nacimiento
- c_pos_final(3) TYPE n VALUE 96, " Posición final
- c_ancho_total(3) TYPE n VALUE 96, " Ancho total del informe,
- c_pos_titul(3) TYPE n VALUE 1, " Posición título,
- c_pos_usuar(3) TYPE n VALUE 37, " Posición usuario
- c_pos_fecha(3) TYPE n VALUE 60, " Posición fecha
- c_pos_pagno(3) TYPE n VALUE 85, " Posición nº de página
- c_pos_nfact(3) TYPE n VALUE 12, " Posición nº factura
- c_pos_fefac(3) TYPE n VALUE 23, " Posición fecha fact
- c_pos_impnt(3) TYPE n VALUE 34, " Posición importe neto
- c_pos_moned(3) TYPE n VALUE 51, " Posición moneda
- c_pos_fifac(3) TYPE n VALUE 57, " Posición final factura.
- c_pos_total(3) TYPE n VALUE 21. " Posición total moneda

* Tablas internas *

* Tabla interna de datos a listar

DATA: BEGIN OF i_zclienxx OCCURS 0.
 INCLUDE STRUCTURE zclienxx.
DATA: END OF i_zclienxx.

* Tabla interna con las facturas por cliente

DATA: BEGIN OF i_factu OCCURS 5,
 moneda LIKE zfactuxx-moned, " Moneda de la factura
 fecha LIKE zfactuxx-fecha, " Fecha
 impnt LIKE zfactuxx-impnt, " Importe neto.

```
nfact LIKE zfactuxx-nfact,      " N° de factura
END OF i_factu.
```

* Pantalla de selección *

* Parámetros

```
SELECTION-SCREEN BEGIN OF BLOCK bloq1 WITH FRAME TITLE text-001.
PARAMETERS p_bukrs LIKE zclienxx-bukrs OBLIGATORY
              DEFAULT '0001'. " Sociedad
SELECTION-SCREEN END OF BLOCK bloq1.
```

* Rangos de selección

```
SELECTION-SCREEN BEGIN OF BLOCK bloq2 WITH FRAME TITLE text-002.
SELECT-OPTIONS:
```

* Añadimos el match-code para el código de cliente

```
s_nclie FOR zclienxx-nclie MATCHCODE OBJECT zcxx, " N° de cliente
s_nombr FOR zclienxx-nombr,      " Nombre cliente
s_fnaci FOR zclienxx-fnaci.      " Fecha de nacimiento
SELECTION-SCREEN END OF BLOCK bloq2.
```

* Comienzo de selección *

START-OF-SELECTION.

* Inicializamos la tabla interna.

```
REFRESH i_zclienxx.
CLEAR i_zclienxx.
```

* Seleccionamos los datos de los clientes que cumplen criterios

```
SELECT *
FROM zclienxx INTO TABLE i_zclienxx
WHERE bukrs = p_bukrs
AND nclie IN s_nclie
AND nombr IN s_nombr
AND fnaci IN s_fnaci.
```

* Comprobamos si no hay datos seleccionados en cuyo caso,

* escribiremos un texto que lo indique.

```
IF NOT ( sy-subrc = 0 ).
WRITE / text-003. " No hay datos para la selección indicada
ENDIF.
```

* Final de la selección *

END-OF-SELECTION.

* Vamos a escribir el listado con los clientes seleccionados en la

* tabla interna I_ZCLIENXX y para cada uno de ellos sus facturas

```
LOOP AT i_zclienxx.
```

* Escribimos el n° de cliente en color clave, el resto en otro color
FORMAT COLOR COL_KEY.

```
WRITE: AT /c_pos_nclie '|' NO-GAP, i_zclienxx-nclie.
FORMAT COLOR COL_NORMAL.
WRITE: AT c_pos_nombr '|' NO-GAP, i_zclienxx-nombr,
      AT c_pos_apel1 '|' NO-GAP, i_zclienxx-apel1,
      AT c_pos_apel2 '|' NO-GAP, i_zclienxx-apel2,
      AT c_pos_fnaci '|' NO-GAP, i_zclienxx-fnaci,
      AT c_pos_final '|' NO-GAP.
* Inicializamos la tabla interna de facturas
REFRESH i_factu.
CLEAR i_factu.
* Recuperamos las facturas correspondientes al cliente
SELECT moned fecha impnt nfact INTO TABLE i_factu
FROM zfactuux
WHERE bukrs = p_bukrs
AND nclie = i_zclienxx-nclie.
* Comprobamos que se han seleccionado facturas para el cliente
* si no es así pasamos al siguiente
CHECK ( sy-subrc = 0 ).
* Escribimos línea de separación
ULINE AT /c_pos_nclie(c_ancho_total).
* Ordenamos las facturas encontradas por moneda
SORT i_factu BY moneda fecha.
* Para cada una de las facturas encontradas
LOOP AT i_factu.
* Activamos color de detalle
FORMAT COLOR COL_POSITIVE.
* Escribimos el detalle de la factura
WRITE: AT /c_pos_nclie '|' NO-GAP,
      AT c_pos_nfact '|' NO-GAP, i_factu-nfact,
      AT c_pos_fefac '|' NO-GAP, i_factu-fecha,
      AT c_pos_impnt '|' NO-GAP, i_factu-impnt CURRENCY
      i_factu-moneda,
      AT c_pos_moned '|' NO-GAP, i_factu-moneda,
      AT c_pos_fifac '|' NO-GAP,
      AT c_pos_final '|' NO-GAP.
* Si es la última entrada para la moneda actual
AT END OF moneda.
* Sumamos totales importes
SUM.
* Línea de separación
ULINE AT /c_pos_nclie(c_ancho_total).
* Activamos el color de totales
FORMAT COLOR COL_TOTAL.
WRITE: AT /c_pos_nclie '|' NO-GAP,
      AT c_pos_total text-013,
      i_factu-impnt CURRENCY i_factu-moneda NO-GAP,
      i_factu-moneda,
      AT c_pos_final '|' NO-GAP.
```

```
* Línea de separación
  ULINE AT /c_pos_nclie(c_ancho_total).
  ENDAT.
  ENDLOOP.
  ENDLOOP.
* Si se han seleccionado datos, escribimos la línea final.
  IF ( sy-subrc = 0 ).
    ULINE AT /c_pos_nclie(c_ancho_total).
  ENDIF.

*****
* Cabecera de página *
*****
TOP-OF-PAGE.
* Hacemos que la línea permanezca fija en el scroll horizontal
  NEW-LINE NO-SCROLLING.
* Activamos el color de cabecera
  FORMAT COLOR COL_HEADING INTENSIFIED OFF.
* Escribimos información de cabecera resaltando los valores
* Título y sociedad
  WRITE: AT c_pos_titul text-009.
  WRITE p_bukrs INTENSIFIED ON.
* Usuario de creación del informe
  WRITE AT c_pos_usuar text-010.
  WRITE sy-uname INTENSIFIED ON.
* Fecha de creación
  WRITE AT c_pos_fecha text-011.
  WRITE sy-datum INTENSIFIED ON USING EDIT MASK '___/___/____!'.
* Nº de página.
  WRITE AT c_pos_pagno text-012.
  WRITE sy-pagno INTENSIFIED ON.
* Escribimos una línea horizontal
  ULINE AT /c_pos_nclie(c_ancho_total).
* Activamos el color de cabeceras intensificado
  FORMAT COLOR COL_HEADING INTENSIFIED ON.
* Escribimos la cabecera de la columnas separadas por el caracter '|'.
* Hacemos que la columna de nº de cliente sea fija en el scroll horiz.
  WRITE: AT /c_pos_nclie '|' NO-GAP, text-004.
* Hasta la columna hacemos fijo
  SET LEFT SCROLL-BOUNDARY.
* Escribimos el detalle de cabecera de cliente
  WRITE: AT c_pos_nombr '|' NO-GAP, text-005,
        AT c_pos_apel1 '|' NO-GAP, text-006,
        AT c_pos_apel2 '|' NO-GAP, text-007,
        AT c_pos_fnaci '|' NO-GAP, text-008,
        AT c_pos_final '|' NO-GAP.
* Escribimos una línea horizontal
  ULINE AT /c_pos_nclie(c_ancho_total).
```

* **Escribimos el detalle de cabecera de facturas**

```
WRITE: AT /c_pos_nclie '|' NO-GAP,
       AT c_pos_nfact '|' NO-GAP, text-014,
       AT c_pos_fefac '|' NO-GAP, text-015,
       AT c_pos_impnt '|' NO-GAP, text-016,
       AT c_pos_moned '|' NO-GAP, text-017,
       AT c_pos_fifac '|' NO-GAP,
       AT c_pos_final '|' NO-GAP.
```

* **Escribimos una línea horizontal**

```
ULINE AT /c_pos_nclie(c_ancho_total).
```

* **Control de la pantalla de selección.**

* **Valores posibles para el rango S_Nombre valor desde.**

```
AT SELECTION-SCREEN ON VALUE-REQUEST FOR s_nombr-low.
```

* **Llamada a la rutina que muestra los valores existentes**

```
PERFORM valores_posibles_nombre USING s_nombr-low.
```

* **Valores posibles para el rango S_Nombre valor hasta**

```
AT SELECTION-SCREEN ON VALUE-REQUEST FOR s_nombr-high.
```

* **Llamada a la rutina que muestra los valores existentes**

```
PERFORM valores_posibles_nombre USING s_nombr-high.
```

* **Rutinas adicionales.**

```
*&-----*
```

```
*& Form VALORES_POSIBLES_NOMBRE
```

```
*&-----*
```

```
* Muestra una ventana con los distintos nombres existentes en *
* la tabla maestro de clientes ZCLIENXX. Dando la posibilidad de *
* seleccionar uno de ellos. Devuelve en el parámetro PS_NOMBR el *
* nombre seleccionado o vacío si no se realiza selección. *
```

```
* -->PS_NOMBR Nombre seleccionado *
```

```
FORM valores_posibles_nombre USING ps_nombr LIKE zclienxx-nombr.
```

* **Variables locales**

* **Tabla interna para almacenar los campos y atributos a visualizar.**

```
DATA : BEGIN OF i_fields OCCURS 1.
        INCLUDE STRUCTURE help_value.
DATA: END OF i_fields.
```

* **Tabla interna para almacenar el valor de campos a mostrar.**

```
DATA : BEGIN OF i_values OCCURS 0,
        valor(20) TYPE c,
```

END OF i_values.

* Proceso *

* Insertamos en la tabla I_FIELDS los atributos del campo a mostrar

CLEAR i_fields.

i_fields-tabname = 'ZCLIENXX'. " Nombre de la tabla

i_fields-fieldname = 'NOMBR'. " Nombre del campo

i_fields-selectflag = 'X'. " Valor seleccionable

APPEND i_fields.

* Seleccionamos los distintos nombres existentes en la tabla ZCLIENXX

* y los almacenamos en la tabla de valores

SELECT DISTINCT nombr INTO zlienxx-nombr

FROM zlienxx

WHERE bukrs = p_bukrs.

i_values-valor = zlienxx-nombr.

APPEND i_values.

ENDSELECT.

* Llamamos a la función predefinida que muestra los datos

CALL FUNCTION 'HELP_VALUES_GET_WITH_TABLE'

* EXPORTING

* CUCOL = 0

* CUROW = 0

* DISPLAY = ''

* FIELDNAME = ''

* TABNAME = ''

* NO_MARKING_OF_CHECKVALUE = ''

* TITLE_IN_VALUES_LIST = ''

* TITEL = ''

* SHOW_ALL_VALUES_AT_FIRST_TIME = ''

* NO_CONVERSION = ''

IMPORTING

select_value = ps_nombr

TABLES

fields = i_fields

valuetab = i_values

EXCEPTIONS

field_not_in_ddic = 1

more_than_one_selectfield = 2

no_selectfield = 3

OTHERS = 4.

ENDFORM. " VALORES_POSIBLES_NOMBRE

El Aspecto del listado es el siguiente:

Listado de maestro de clientes
 Lista Tratar Pasar a Sistema Ayuda

Listado de clientes sociedad: 0001 Usuario: ACCRICOTE Fecha: 05/12/2001 Pag: 1

N.Cliente	Nombre	Primer Apellido	Segundo apellido	F.Nacim.
000000001	PEDRO	PÉREZ	LÓPEZ	04.12.2000
	000000003	04.12.2001	2,02	ESP
	000000002	04.12.2001	20,000	ESP
Total moneda:			20,202	ESP
	000000001	04.12.2001	145,05	EUR
Total moneda:			145,05	EUR
000000002	JORGE	SANCHEZ		01.02.1999
	000000004	04.12.2001	2,02	EUR
Total moneda:			2,02	EUR

4.2.10 Bases de datos lógicas.

Una base de datos lógica (**LDB**) proporciona una visión lógica de las tablas físicas, pudiendo relacionar tablas entre sí. Las bases de datos lógicas (LDB) son herramientas en los que se definen los accesos a la información almacenada en las tablas del diccionario así como las relaciones entre ellas, proporcionan por tanto, una visión lógica de las tablas. Están formadas por un conjunto de programas en los que se detallan las tablas utilizadas, las relaciones entre ellas, los parámetros de acceso....

Las LDB simplifican la programación de Reports ya que dentro de su lógica se pueden encapsular los accesos de lectura, verificación de autorizaciones, parámetros de selección...etc. Por tanto se suelen utilizar si existen un conjunto elevado de programas que utilizan la misma información de base de datos.

El principal inconveniente de la bases de datos lógica es su no especialización, es decir, las selecciones a las bases de datos son generales y por tanto serán normalmente más lentas que las selecciones específicas que se pueden realizar dentro de los reports.

Podemos crear nuestras propias bases de datos lógicas a través del menú 'Herramientas → Workbench ABAP → Desarrollo → Entorno de programación → Bases de datos lógicas' (SE36). La creación de una base de datos lógica pasa por definir:

- Las tablas utilizadas y su relación.
- Los parámetros de selección sobre las tablas y sus textos. Con sus respectivos match-code si proceden.
- La codificación de las selecciones a las tablas dentro de los procedimientos PUT que serán invocados cuando se ejecute la instrucción de lectura de LDB 'GET'.
- La documentación de la LDB.

A su vez, existen s bases de datos lógicas predefinidas que podemos utilizar en nuestros programas, vamos a ver una de ellas. La estructura de la base de datos lógica 'KDF' es la siguiente:

- **Evento GET.**
GET permite obtener los datos de las tablas. El evento GET 'tablan' llamará al procedimiento PUT 'tablan' codificado en el programa de la base de datos lógica.

La estructura de acceso a los datos desde los reports que utilicen esta base de datos lógica será algo así:

```
GET LFA1.  
Sentencias.  
GET LFB1.  
Sentencias.  
GET BSIK.  
.....
```

El funcionamiento sería el siguiente, para cada uno de los registros encontrados en la tabla LFA1 (que cumplan los criterios de selección), se procesarían los registros de la tabla LFB1 y con cada uno de estos los de la tabla BSIK y así sucesivamente. Tiene dos opciones:

LATE: Hace que esta instrucción se ejecute una vez procesados todos los registros.

FIELDS: Hace que solamente se recoja la información de los campos que se indican a continuación.

Si quisiéramos hacer un listado con las partidas abiertas (facturas pendientes de ser pagadas) de cada acreedor bastaría realizar un REPORT en el que indicásemos que utiliza la base de datos lógica KDF. Para ello en la pantalla de atributos del programa indicaremos 'KD' en el campo 'Base de datos lógica' y 'F' en el campo 'de la aplicación'.

El Report podría ser algo así:

```
*****  
* PROGRAMA: ZREPO1XX. *  
* DESCRIPCION: Muestra un listado de las facturas pendientes *
```

```

*           de pago de los acreedores existentes.
* AUTOR : MAD00           FECHA: 20/08/2001
*-----*
* CONTROL DE MODIFICACIONES
* FECHA.   AUTOR.   DESCRIPCION MODIFICACION.
*****
REPORT zrepo1xx LINE-SIZE 90.
*****
* Tablas de diccionario de datos
*****
TABLES: t001,           " Maestro de sociedades
        lfa1,           " Maestro acreedores general
        lfb1,           " Maestro acreedores sociedad
        bsik.           " Partidas abiertas acreedor
*****
* Definición de variables globales
*****
DATA :d_dmbtr LIKE bsik-dmbtr.           " Total sociedad

*****
* Evento de comienzo de selección.
*****
START-OF-SELECTION.
* Obtenemos los datos de acreedores
GET lfa1.
  FORMAT COLOR COL_KEY.
* Escribimos nº y nombre del acreedor
  WRITE: / lfa1-lifnr,           " Nº de acreedor
         lfa1-name1.           " Nombre
* Obtenemos datos de sociedad
GET lfb1.
* Obtenemos la moneda de la sociedad
  SELECT SINGLE waers INTO (t001-waers)
  FROM t001
  WHERE bukrs = lfb1-bukrs.
* Obtenemos los datos de las partidas abiertas de los acreedores.
GET bsik.
  FORMAT COLOR COL_NORMAL.
* Obtenemos el signo en función del campo debe/haber
  IF ( bsik-shkzg = 'S' ).
    bsik-wrbtr = bsik-wrbtr * -1.
    bsik-dmbtr = bsik-dmbtr * -1.
  ENDIF.
* Escribimos los valores

  WRITE: / bsik-bukrs,           " Sociedad
         bsik-belnr,           " Nº doc.
         bsik-zfbdt,           " Fecha vencimiento

```

bsik-wrbtr CURRENCY bsik-waers, "Importe en mon. doc.
 bsik-waers, " Moneda doc.
 bsik-dmbtr CURRENCY t001-waers, " Importe moneda loc.

t001-waers. " Moneda local.

* **Acumulamos el importe en moneda local**

d_dmbtr = d_dmbtr + bsik-dmbtr.

* **Cuando finaliza el tratamiento de cada sociedad**

GET lfb1 LATE.

FORMAT COLOR COL_TOTAL.

* **Escribimos el importe en moneda de la sociedad del saldo**

* **'Importe total para la sociedad : XXXXX XXX'.**

WRITE: text-001, d_dmbtr CURRENCY t001-waers, t001-waers.

* **Fin de línea e inicialización de importe total sociedad**

ULINE /.

CLEAR d_dmbtr.

Definiendo las cabeceras y textos oportunos, tendremos un listado con el siguiente aspecto:

The screenshot shows a window titled 'Listado de maestro de clientes' with a menu bar (Lista, Tratar, Pasar a, Sistema, Ayuda) and a toolbar. The main content area displays a table of open items under the heading 'Partidas abiertas'. The table has columns for 'Nº Acre.', 'Nombre Acreedor', 'Soc.', 'Documento', 'F.Uenci.', 'Importe MD', 'MD', 'Importe ML', and 'ML'. The data rows show various account numbers and amounts in EUR. The final row is highlighted in yellow and reads 'Importe total para la sociedad : 120,20- EUR'.

Nº Acre.	Nombre Acreedor	Soc.	Documento	F.Uenci.	Importe MD	MD	Importe ML	ML
EX0910459G	XXXXXXXXXX XXXXXXXXXXXXXXXX							
1100	3000000001	26.02.2001			6.010,12	EUR	6.010,12	EUR
1100	1000000832	23.03.2001			6.130,32-	EUR	6.130,32-	EUR
1100	1000000746	20.03.2001			154.103	ESP	926,18	EUR
1100	1000000650	01.01.2001			93.966-	ESP	564,75-	EUR
1100	1000000649	06.02.2001			7.598-	ESP	45,66-	EUR
1100	1000000648	03.01.2001			52.539-	ESP	315,77-	EUR
1100	3000000000	21.02.2001			6.010,12	EUR	6.010,12	EUR
1100	1000000471	06.02.2001			7.598-	ESP	45,66-	EUR
1100	1000000649	06.02.2001			7.598	ESP	45,66	EUR
1100	2100000034	06.02.2001			7.598-	ESP	45,66-	EUR
1100	2100000035	06.02.2001			7.598	ESP	45,66	EUR
1100	1000000572	01.01.2001			93.966-	ESP	564,75-	EUR
1100	1000000650	01.01.2001			93.966	ESP	564,75	EUR
1100	1000000465	03.01.2001			52.539-	ESP	315,77-	EUR
1100	1000000648	03.01.2001			52.539	ESP	315,77	EUR
1100	2100000021	21.02.2001			6.010,12-	EUR	6.010,12-	EUR
Importe total para la sociedad :					120,20-	EUR		

4.2.11 FIELD-GROUPS.

Como su nombre indica, representa un conjunto de campos. Al definir un conjunto de campos, únicamente se define el nombre del grupo, para incluir campos en el conjunto será necesario utilizar la instrucción INSERT.

Son útiles cuando hay que mostrar varias líneas de diferente tipo dentro de un mismo informe, por ejemplo en caso anterior, mostramos líneas con datos de los proveedores junto con sus documentos.

Relacionados con los Field-groups tenemos las siguientes instrucciones:

- INSERT Permite añadir un campo a un conjunto de campos.
- EXTRACT Permite volcar la información a los campos que forman el conjunto. Formando un 'registro' con los campos de los que se compone el conjunto. Cuando se define el conjunto especial HEADER, a cada uno de estos 'registros' se le añaden los campos contenidos en él, formando un 'registro' compuesto por los campos HEADER más los del conjunto.
- LOOP - ENDLOOP.
 - Con los ya conocidos eventos:
 - AT NEW f
 - AT END F
 - AT FIST
 - AT LAST
 - Y el nuevo:
 - AT fg Donde fg sería un conjunto de campos, de tal modo que este evento siempre se ejecutará cuando el tipo de registro actual del loop coincida con un tipo de registro definido por los campos que componen el conjunto.

Vamos a generar un Report, que genera el mismo listado que en el ejemplo anterior pero ayudándonos de FIELD-GROUPS.

Vamos a definir tres conjuntos de campos,

HEADER: Con los campos LFA1-LIFNR, BSIK-BUKRS y BSIK-BELNR.

DIRECCION: Con los campos LFA1-NAME1, LFA1-STRAS, LFA1-PSTLZ y LFA1-ORT01.

PARTIDAS: BSIK-DMBTR, BSIK-WRBTR, BSIK-SHKZG, BSIK-WAERS.

Definir el HEADER implicará que los tipos de registros de DIRECCION estén formados por los campos:

LFA1-LIFNR, BSIK-BUKRS, BSIK-BELNR, LFA1-NAME1, LFA1-STRAS, LFA1-PSTLZ, LFA1-ORT01.

Y los registros PARTIDAS estén compuestos por:

LFA1-LIFNR, BSIK-BUKRS, BSIK-BELNR, BSIK-DMBTR BSIK-WRBTR BSIK-SHKZG BSIK-WAERS.

De esta manera los campos del HEADER se convierten en campos claves para la ordenación.

El código será algo así

```
*****
* PROGRAMA: ZREPO2XX. *
* DESCRIPCION: Muestra un listado de las facturas pendientes *
* de pago de los acreedores existentes. *
* AUTOR : MAD00 FECHA: 20/08/2001 *
* -----*
* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *
*****
REPORT zrepo2xx LINE-SIZE 75.
*****
* Tablas de diccionario de datos
*****
TABLES: t001, " Maestro de sociedades
        lfa1, " Maestro acreedores general
        lfb1, " Maestro acreedores sociedad
        bsik. " Partidas abiertas acreedor

*****
* Definición de variables globales *
*****
DATA :d_dmbtr LIKE bsik-dmbtr. " Total sociedad

*****
* Definición de FIELD-GROUPS *
*****

* Definimos los conjuntos de campos
FIELD-GROUPS : header, direccion, partidas.

* Incluimos los campos en cada uno de los conjuntos
INSERT lfa1-lifnr bsik-bukrs bsik-belnr INTO header.
INSERT lfa1-name1 lfa1-stras lfa1-pstlz lfa1-ort01 INTO direccion.
INSERT bsik-dmbtr bsik-wrbtr bsik-shkzg bsik-waers INTO partidas.

*****
* Evento de comienzo de selección.
*****
START-OF-SELECTION.
```

GET lfa1.

- * Obtenemos los datos de los campos del conjunto dirección
EXTRACT direccion.

GET bsik.

- * Obtenemos los datos de los campos del conjunto partidas
EXTRACT partidas.

* Evento de comienzo final de selección

END-OF-SELECTION.

- * Ordenamos, será por los campos de la cabecera, es decir,
* los campos del conjunto HEADER.

SORT.

- * Recooremos todos los registros seleccionados
LOOP.

- * Para los registros de tipo dirección
AT direccion.

- * Activamos color
FORMAT COLOR COL_KEY.

- * Escribimos datos del acreedor
WRITE: lfa1-lifnr,lfa1-name1, lfa1-stras, lfa1-pstlz, lfa1-ort01.

ENDAT.

- * Para cada nueva sociedad
AT NEW bsik-bukrs.

- * Seleccionamos la moneda de la sociedad para totalizar.
SELECT SINGLE waers INTO (t001-waers)
FROM t001
WHERE bukrs = bsik-bukrs.

ENDAT.

- * Para cada tipo de registro de documentos
AT partidas.

FORMAT COLOR COL_NORMAL.

- * **Obtenemos el signo en función del campo debe/haber**
IF (bsik-shkzg = 'S').

```
bsik-wrbtr = bsik-wrbtr * -1.  
bsik-dmbtr = bsik-dmbtr * -1.
```

ENDIF.

- * **Escribimos los valores**

```
WRITE: / bsik-bukrs,           " Sociedad  
       bsik-belnr,           " N° doc.  
       bsik-zfbdt,           " Fecha vencimiento  
       bsik-wrbtr CURRENCY bsik-waers, "Importe en mon. doc.  
       bsik-waers,           " Moneda doc.  
       bsik-dmbtr CURRENCY t001-waers, " Importe mon soc  
       t001-waers.           " Moneda local.
```

- * **Acumulamos el importe en moneda local**

```
d_dmbtr = d_dmbtr + bsik-dmbtr.  
ENDAT.
```

- * **En la finalización de cada sociedad**

```
AT END OF bsik-bukrs.  
  FORMAT COLOR COL_TOTAL.
```

- * **Escribimos total para la sociedad.**

```
WRITE: / text-001,  
       bsik-bukrs, d_dmbtr CURRENCY t001-waers,  
       t001-waers.  
ULINE /.
```

- * **Inicializamos el total**

```
CLEAR d_dmbtr.  
ENDAT.  
ENDLOOP.
```

(La salida de este programa es idéntica al anterior, salvo que se muestran más datos personales del acreedor, dirección...).

4.3 Programación de listados interactivos

4.3.1 Introducción.

Como hemos visto en el apartado anterior, mediante los listados planos podemos mostrar información sin que el usuario actúe sobre el informe, únicamente se limita a indicar los parámetros de selección. Además de mostrar esta información podemos hacer que el usuario tenga opciones sobre el informe

para introducir datos, desglosar la información, seleccionar un conjunto de registros.... Todo esto se realiza dando al informe opciones de usuario.

4.3.2 Eventos

Para manejar las opciones de usuario, disponemos de varios eventos en los cuales podemos codificar cada una de las funciones del listado.

- **AT LINE-SELECTION**

Este evento se ejecuta cada vez que se selecciona una línea de la pantalla, para seleccionar una línea de un listado, existen varias formas, las más habituales son hacer Doble-click sobre la línea, hacer Click sobre la línea cuando nos aparece el cursor con la forma de *mano*  o situados sobre la línea pulsar una tecla de función generalmente (F2).

- **AT USER-COMMAND**

Este evento se ejecuta cada vez que el usuario pulsa una opción, bien a través de un menú, bien a través de un botón...

Cada opción de usuario, lleva implícito un código de comando, cuando se selecciona una opción, en la variable del sistema SY-UCOMM se almacena el código de comando asociado. Dentro de este evento, realizaremos tantos tratamientos como opciones tengamos en el listado. Normalmente el esquema de programación de este evento es algo así:

```
CASE SY-UCOMM.  
  WHEN 'OPC1'.  
    * Tratamiento de la opción 1  
  WHEN 'OPC2'.  
    * Tratamiento de la opción 2.  
  ....  
  WHEN 'OPCn'.  
    * Tratamiento de la opción n.
```

Hay opciones que ya están predefinidas en un informe, por ejemplo las opción de seleccionar una línea del evento AT LINE-SELECTION que tiene el valor PICK, las opciones de paginar (P+, P-, P++ y P--). ...

- **AT PFn**

'n' representa un número entre 0 y 99, por tanto podemos tener hasta 100 eventos distintos cada uno 'gobernará' la tecla de función correspondiente.

Este evento se ejecuta cada vez que se ejecute la tecla de función n. Normalmente este evento no se utiliza ya que las teclas de función van asociadas a comandos que se pueden controlar en el evento AT USER-COMMAND.

4.3.3 Sentencias de lectura y escritura de líneas

Para poder capturar la información de la línea del listado seleccionada, tenemos las instrucciones:

- **HIDE:**

Permite almacenar el valor de un campo en relación con la línea de escritura, debe de indicarse a continuación de la sentencia WRITE. De esta forma al seleccionar la línea, directamente el campo tomará el valor que tenga en la línea. Para almacenar el valor de un campo con HIDE, no es necesario escribir el campo previamente con la sentencia WRITE, es decir, se pueden almacenar campos que no se escriban en la línea del listado.

Por ejemplo:

```
WRITE D_NCLIENTE, D_NOMBRE.  
HIDE D_NCLIENTE.
```

Al seleccionar la línea, en la variable D_NCLIENTE tendremos el valor del N° de cliente.

- **READ LINE...**

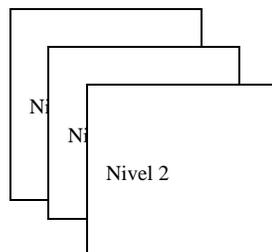
Lee el contenido de una línea del informe y lo almacena sobre los campos indicados. Tiene las siguientes variantes:

```
READ LINE lin.  
READ LINE lin OF CURRENT PAGE.  
READ LINE lin OF PAGE pag.  
READ CURRENT LINE.
```

Para indicar los campos que desean ser leídos se añade FIELD VALUE f1 into g1 f2 into g2 . (Si el campo destino coincide con el origen, se puede abreviar con FIELD VALUE fi).

4.3.4 Niveles de listados anidados.

Cada vez que se produce un evento de usuario y a continuación, se escribe información en la pantalla, se produce un nuevo nivel de lista, es decir, 'se escribe sobre una nueva hoja que está sobre la anterior', la hoja anterior permanece y podremos volver a ella.



El nivel de lista se controla con la variable SY-LSIND.

De tal forma que esta será incrementando después de cada opción de usuario, si tras un evento de usuario queremos permanecer en el mismo nivel deberemos indicarlo decrementando en una esta variable. ($SY-LSIND = SY-LSIND - 1$). Otra variable relacionada con el nivel de lista es la variable SY-LISTI, que indica el nivel de lista actual, normalmente vale $SY-LSIND - 1$.

4.3.5 Ejemplo de listado interactivo I.

Vamos a crear un listado interactivo, partiendo del listado de clientes, programa 'ZREPORXX', haremos una copia de este programa y lo llamaremos 'ZREPO3XX'.

La primera modificación consistirá en escribir el campo número de cliente con la opción HOTSPOT para que sea sensible a un Click. Además almacenaremos el valor con HIDE para que se recupere automáticamente el valor del campo del cliente seleccionado. Con el cliente seleccionado, vamos a mostrar un mensaje.

Para ello:

Dentro del evento END-OF-SELECTION.

Modificaremos la línea donde escribimos el código de cliente para activar la selección con Click sobre este campo.

* **Escribimos en modo Click.**

```
WRITE: AT /C_POS_NCLIE '|' NO-GAP, I_ZCLIENXX-NCLIE HOTSPOT.
```

* **Almacenamos el valor del campo**

```
HIDE I_ZCLIENXX-NCLIE.
```

Después de escribir todos los clientes, inicializamos la cabecera de la tabla I_ZCLIENXX.

```
CLEAR I_ZCLIENXX.
```

Añadimos el evento de selección de líneas con el siguiente código.

```
AT LINE-SELECTION.
```

* **Comprobamos si es una línea válida, es decir el campo está informado**

```
IF ( I_ZCLIENXX-NCLIE IS INITIAL ).
```

```
MESSAGE E000(38) WITH TEXT-013.
```

* **Si se ha seleccionado una línea válida.**

```
ELSE.
```

```
MESSAGE I000(38) WITH TEXT -014 I_ZCLIENXX-NCLIE.
```

```
ENDIF.
```

* **Inicializamos el contenido del campo.**

```
CLEAR I_ZCLIENXX-NCLIE.
```

Utilizamos la clase de mensajes 38 y el mensaje 000 que no tiene texto, solamente parámetros. Como parámetros pasamos los textos correspondientes que son:

TEXT-013 -- 'Línea no válida'.

TEXT-014 -- 'Cliente seleccionado: '.

Podemos comprobar como con un Click sobre el campo NCLIE se muestra el mensaje, pero se necesita un Doble-click para conseguir el mismo efecto en el resto de la línea.

Vamos a complicar nuestro listado de tal forma que, en lugar de mostrar un mensaje con el cliente seleccionado, muestre en un nivel superior otro listado con las facturas del cliente.

Para ello tendremos que realizar las siguientes modificaciones:

Dentro del evento AT LINE-SELECTION.

- * [Comprobamos si es una línea válida, es decir el campo está informado](#)
IF (I_ZCLIENXX-NCLIE IS INITIAL).
MESSAGE E000(38) WITH TEXT-013.
- * [Si se ha seleccionado una línea válida.](#)
ELSE.
- * [Llamamos al procedimiento para mostrar las facturas del cliente.](#)
PERFORM MOSTRAR_FACTURAS_CLIENTE USING I_ZCLIENXX-NCLIE.
ENDIF.
- * [Inicializamos el contenido del campo.](#)
CLEAR I_ZCLIENXX-NCLIE.

Por otra parte será necesario añadir un nuevo evento TOP-OF-PAGE DURING LINE-SELECTION para escribir la cabecera de página del detalle de las facturas.

TOP-OF-PAGE DURING LINE-SELECTION.

- * [Escribimos una línea horizontal](#)
ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
- * [Activamos color de cabecera](#)
FORMAT COLOR COL_HEADING.
- * [Escribimos el detalle de la cabecera de factura.](#)
WRITE: AT /C_POS_NFACT '|' NO-GAP, TEXT-017,
AT C_POS_FEFAC '|' NO-GAP, TEXT-018,
AT C_POS_MESFA '|' NO-GAP, TEXT-019,
AT C_POS_IMPNT '|' NO-GAP, TEXT-020,
AT C_POS_MONED '|' NO-GAP, TEXT-021,
AT C_POS_FIFAC '|' NO-GAP.
- * [Escribimos una línea horizontal](#)
ULINE AT /C_POS_NFACT(C_ANCHO_FAC).

El código correspondiente al procedimiento MOSTRAR_FACTURAS_CLIENTE será algo así:

```

*&-----*
*&   Form MOSTRAR_FACTURAS_CLIENTE
*&-----*
*   Selecciona las facturas correspondientes al cliente recibido *
*   como parámetro y las muestra en pantalla. *
*   Si no se seleccionan facturas, se muestra el mensaje correspondiente
*-----*
*   -->PE_CLIENX  -- N° de cliente. *
*-----*
FORM   MOSTRAR_FACTURAS_CLIENTE   USING   VALUE(PE_NCLIE)   LIKE
ZCLIENXX-NCLIE.

*****
* Variables locales *
*****
* Tabla interna para almacenar las facturas del cliente
DATA: BEGIN OF I_ZFACTUXX OCCURS 0.
      INCLUDE STRUCTURE ZFACTUXX.
DATA: END OF I_ZFACTUXX.
*****
* Proceso *
*****

* Seleccionamos las facturas del cliente para la sociedad indicada
SELECT *
FROM ZFACTUXX INTO TABLE I_ZFACTUXX
WHERE BUKRS = P_BUKRS
AND   NCLIE = PE_NCLIE.
* Si se han seleccionado facturas
IF ( SY-SUBRC = 0 ).
* Activamos color de detalle
  FORMAT COLOR COL_NORMAL.

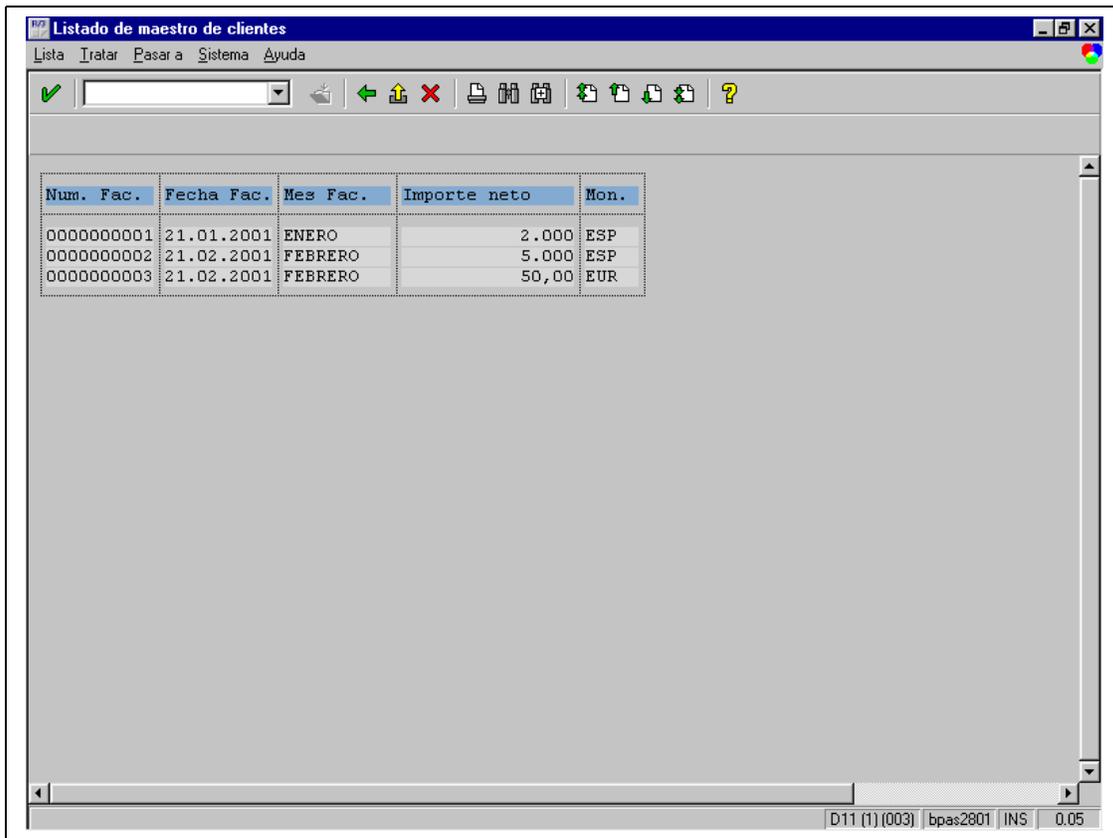
* Para cada una de las facturas seleccionadas
  LOOP AT I_ZFACTUXX.
* Escribimos los datos de las facturas.
WRITE: AT /C_POS_NFACT '|' NO-GAP, I_ZFACTUXX-NFACT,
      AT C_POS_FEFAC '|' NO-GAP, I_ZFACTUXX-FECHA,
      AT C_POS_MESFA '|' NO-GAP, I_ZFACTUXX-MES,
      AT C_POS_IMPNT '|' NO-GAP, I_ZFACTUXX-IMPNT CURRENCY
      I_ZFACTUXX-MONEDA,
      AT C_POS_MONED '|' NO-GAP, I_ZFACTUXX-MONEDA,
      AT C_POS_FIFAC '|' NO-GAP.
  ENDLOOP.
* Escribimos línea final
  ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
* Si no se han seleccionado
ELSE.

```

Curso programación ABAP IV

```
* Mostramos mensaje de información con
* 'No existen facturas para el cliente:'xxx 'para la sociedad:' xxx
MESSAGE I000(38) WITH TEXT-015 PE_NCLIE TEXT-016 P_BUKRS.
ENDIF.
ENDFORM.          " MOSTRAR_FACTURAS_CLIENTE
```

El listado con las facturas del cliente tendrá el siguiente aspecto:



The screenshot shows a SAP window titled 'Listado de maestro de clientes'. The window contains a table with the following data:

Num. Fac.	Fecha Fac.	Mes Fac.	Importe neto	Mon.
0000000001	21.01.2001	ENERO	2.000	ESP
0000000002	21.02.2001	FEBRERO	5.000	ESP
0000000003	21.02.2001	FEBRERO	50,00	EUR

El siguiente paso a realizar va a consistir en mostrar el detalle de una factura, cuando sea seleccionada en el listado, mostrando los datos de la factura, del cliente y calculando el IVA correspondiente (Suponemos valor fijo 16%). El formato de impresión debe ser:

FACTURA Nº xxxxxxxxxx
Cliente: xx
Fecha: dd.mm.aaaa

```
Mes facturación: xxxxxxxxxx
Moneda: xxxxx
    Importe Neto : xxxxxxxxxxxxxxxxx
    Importe IVA  : xxxxxxxxxxxxxxxxx
    Total Factura: xxxxxxxxxxxxxxxxx
```

Para conseguir esto realizaremos las siguientes modificaciones en el programa:

Dentro del evento AT-LINE SELECTION será necesario distinguir el nivel de detalle en el que nos encontramos para actuar de un modo u otro, es decir, mostrando la lista de facturas, si estamos en el nivel 0 o mostrando el detalle de la factura si estamos en el nivel 1. Mediante la variable del sistema SY-LISTI podemos saber en todo momento en que nivel de lista nos encontramos.

Del mismo modo tendremos que controlar en el evento TOP-OF-PAGE DURING LINE-SELECTION para escribir o no la cabecera de listado, ya que en el detalle de factura, no vamos a escribir ninguna cabecera de página.

Otra modificación importante que debemos realizar es hacer global la tabla interna de facturas I_ZFACTUXX, hasta ahora local al procedimiento, MOSTRAR_FACTURAS_CLIENTE, de este modo podremos almacenar su valor con HIDE inmediatamente después de su escritura y este podrá ser recuperado con posterioridad en el evento AT LINE-SELECTION.

Finalmente crearemos un nuevo procedimiento MOSTRAR_DETALLE_FACTURA para escribir el detalle de la factura seleccionada, recuperando los datos de la factura del cliente correspondiente y calculando los importes de IVA y Total factura.

El código del programa resultante se muestra a continuación:

```
*****
* PROGRAMA: ZREPO3XX. *
* DESCRIPCION: Muestra un listado de los clientes existentes en el *
* que se detalla en nº de cliente junto con el nombre y *
* apellidos. Así mismo se podrán visualizar las facturas *
* de un cliente y también el detalle de las mismas. *
* AUTOR: MAD00 FECHA: 13/08/2001 *
*-----*
* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *
*****
REPORT ZREPO3XX NO STANDARD PAGE HEADING LINE-SIZE 120 LINE-COUNT
80.
```

* Tablas del diccionario de datos *

TABLES: ZCLIENXX, " Maestro de clientes.
ZFACTUXX. " Facturas clientes

* Definición de constantes *

CONSTANTS:

* Constantes posiciones clientes

C_POS_NCLIE(3) TYPE N VALUE 1, " Posición nº cliente
C_POS_NOMBR(3) TYPE N VALUE 12, " Posición nombre
C_POS_APEL1(3) TYPE N VALUE 33, " Posición apellido1
C_POS_APEL2(3) TYPE N VALUE 59, " Posición apellido2
C_POS_FNACI(3) TYPE N VALUE 85, " Posición F. Nacimiento
C_POS_FINAL(3) TYPE N VALUE 96, " Posición final
C_ANCHO_TOTAL(3) TYPE N VALUE 96, " Ancho total del informe,
C_POS_TITUL(3) TYPE N VALUE 1, " Posición título,
C_POS_USUAR(3) TYPE N VALUE 37, " Posición usuario
C_POS_FECHA(3) TYPE N VALUE 60, " Posición fecha
C_POS_PAGNO(3) TYPE N VALUE 85, " Posición nº de página

* Constantes posiciones facturas.

C_POS_NFACT(3) TYPE N VALUE 1, " Posición Nº de factura
C_POS_FEFAC(3) TYPE N VALUE 12, " Posición Fecha
C_POS_MESFA(3) TYPE N VALUE 23, " Posición Mes
C_POS_IMPNT(3) TYPE N VALUE 34, " Posición Importe
C_POS_MONED(3) TYPE N VALUE 51, " Posición moneda
C_POS_FIFAC(3) TYPE N VALUE 57, " Posición final factura.
C_ANCHO_FAC(3) TYPE N VALUE 57, " Ancho detalle factura.

* Constantes de posiciones de detalle de factura

C_POS_D_INICI(3) TYPE N VALUE 1, " Posición inicial
C_POS_D_FINAL(3) TYPE N VALUE 80, " Posición final
C_POS_D_NFACT(3) TYPE N VALUE 20, " Posición nfactura
C_POS_D_IMPOR(3) TYPE N VALUE 20. " Posición importes

* Tablas internas *

* Tabla interna de datos a listar

DATA: BEGIN OF I_ZCLIENXX OCCURS 0.
INCLUDE STRUCTURE ZCLIENXX.
DATA: END OF I_ZCLIENXX.

* Tabla interna con datos de facturas

DATA: BEGIN OF I_ZFACTUXX OCCURS 0.
INCLUDE STRUCTURE ZFACTUXX.

DATA: END OF I_ZFACTUXX.

```
*****
* Pantalla de selección *
*****
* Parámetros
SELECTION-SCREEN BEGIN OF BLOCK BLOQ1 WITH FRAME TITLE TEXT-001.
PARAMETERS P_BUKRS LIKE ZCLIENXX-BUKRS OBLIGATORY
           DEFAULT '0001'. " Sociedad
SELECTION-SCREEN END OF BLOCK BLOQ1.
* Rangos de selección
SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE TEXT-002.
SELECT-OPTIONS:
* Añadimos el match-code para el código de cliente
  S_NCLIE FOR ZCLIENXX-NCLIE MATCHCODE OBJECT ZCXX, " N° de cliente
  S_NOMBR FOR ZCLIENXX-NOMBR, " Nombre cliente
  S_FNACI FOR ZCLIENXX-FNACI. " Fecha de nacimiento
SELECTION-SCREEN END OF BLOCK BLOQ2.
```

```
*****
* Comienzo de selección *
*****
START-OF-SELECTION.
* Inicializamos la tabla interna.
REFRESH I_ZCLIENXX.
CLEAR I_ZCLIENXX.
* Seleccionamos los datos de los clientes que cumplen criterios
SELECT *
FROM ZCLIENXX INTO TABLE I_ZCLIENXX
WHERE BUKRS = P_BUKRS
AND NCLIE IN S_NCLIE
AND NOMBR IN S_NOMBR
AND FNACI IN S_FNACI.
* Comprobamos si no hay datos seleccionados en cuyo caso,
* escribiremos un texto que lo indique.
IF NOT ( SY-SUBRC = 0 ).
WRITE / TEXT-003. " No hay datos para la selección indicada
ENDIF.
```

```
*****
* Final de la selección *
*****
END-OF-SELECTION.
* Vamos a escribir el listado con los clientes seleccionados en la
* tabla interna I_ZCLIENXX
LOOP AT I_ZCLIENXX.
* Escribimos el n° de cliente en color clave, el resto en otro color
FORMAT COLOR COL_KEY.
* Escribimos en modo Click.
```

WRITE: AT /C_POS_NCLIE '|' NO-GAP, I_ZCLIENXX-NCLIE HOTSPOT.

* Almacenamos el valor del campo

HIDE I_ZCLIENXX-NCLIE.

FORMAT COLOR COL_NORMAL.

WRITE: AT C_POS_NOMBR '|' NO-GAP, I_ZCLIENXX-NOMBR,
AT C_POS_APEL1 '|' NO-GAP, I_ZCLIENXX-APEL1,
AT C_POS_APEL2 '|' NO-GAP, I_ZCLIENXX-APEL2,
AT C_POS_FNACI '|' NO-GAP, I_ZCLIENXX-FNACI,
AT C_POS_FINAL '|' NO-GAP.

ENDLOOP.

* Si se han seleccionado datos, escribimos la línea final.

IF (SY-SUBRC = 0).

ULINE AT /C_POS_NCLIE(C_ANCHO_TOTAL).

ENDIF.

* Inicializamos el valor de la cabecera de la tabla

CLEAR I_ZCLIENXX.

* Cabecera de página *

TOP-OF-PAGE.

* Hacemos que la línea permanezca fija en el scroll horizontal

NEW-LINE NO-SCROLLING.

* Activamos el color de cabecera

FORMAT COLOR COL_HEADING INTENSIFIED OFF.

* Escribimos información de cabecera resaltando los valores

* Título y sociedad

WRITE: AT C_POS_TITUL TEXT-009.

WRITE P_BUKRS INTENSIFIED ON.

* Usuario de creación del informe

WRITE AT C_POS_USUAR TEXT-010.

WRITE SY-UNAME INTENSIFIED ON.

* Fecha de creación

WRITE AT C_POS_FECHA TEXT-011.

WRITE SY-DATUM INTENSIFIED ON USING EDIT MASK '___/___/___'.

* Nº de página.

WRITE AT C_POS_PAGNO TEXT-012.

WRITE SY-PAGNO INTENSIFIED ON.

* Escribimos una línea horizontal

ULINE AT /C_POS_NCLIE(C_ANCHO_TOTAL).

* Activamos el color de cabeceras intensificado

FORMAT COLOR COL_HEADING INTENSIFIED ON.

* Escribimos la cabecera de la columnas separadas por el caracter '|'.
* Hacemos que la columna de nº de cliente sea fija en el scroll horiz.

WRITE: AT /C_POS_NCLIE '|' NO-GAP, TEXT-004.

* Hasta la columna hacemos fijo

SET LEFT SCROLL-BOUNDARY.

* Escribimos el detalle de cabecera de cliente

```
WRITE: AT C_POS_NOMBR '|' NO-GAP, TEXT-005,  
       AT C_POS_APEL1 '|' NO-GAP, TEXT-006,  
       AT C_POS_APEL2 '|' NO-GAP, TEXT-007,  
       AT C_POS_FNACI '|' NO-GAP, TEXT-008,  
       AT C_POS_FINAL '|' NO-GAP.
```

* Escribimos una línea horizontal

```
ULINE AT /C_POS_NCLIE(C_ANCHO_TOTAL).
```

* Cabecera de página durante la selección *

TOP-OF-PAGE DURING LINE-SELECTION.

* Comprobamos el nivel de listado para determinar la cabecera
CASE SY-LISTI.

* Si nivel 0. Activar cabecera facturas

WHEN 0.

* Escribimos una línea horizontal

```
ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
```

* Activamos color de cabecera

```
FORMAT COLOR COL_HEADING.
```

* Escribimos el detalle de la cabecera de factura.

```
WRITE: AT /C_POS_NFACT '|' NO-GAP, TEXT-017,  
       AT C_POS_FEFAC '|' NO-GAP, TEXT-018,  
       AT C_POS_MESFA '|' NO-GAP, TEXT-019,  
       AT C_POS_IMPNT '|' NO-GAP, TEXT-020,  
       AT C_POS_MONED '|' NO-GAP, TEXT-021,  
       AT C_POS_FIFAC '|' NO-GAP.
```

* Escribimos una línea horizontal

```
ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
```

* Si nivel 1. No escribimos cabecera.

WHEN 1.

ENDCASE.

* Evento de selección de línea *

AT LINE-SELECTION.

* En función del nivel de lista en el que nos encontremos
CASE SY-LISTI.

* Si nivel 0, mostrar facturas cliente

WHEN 0.

* Comprobamos si es una línea válida,

* es decir el campo de código de cliente está informado

```
IF ( I_ZCLIENXX-NCLIE IS INITIAL ).
```

```
MESSAGE E000(38) WITH TEXT-013.
```

* Si se ha seleccionado un línea válida.

```
ELSE.
```

* Llamamos al procedimiento para mostrar las facturas del cliente.

```

PERFORM MOSTRAR_FACTURAS_CLIENTE USING I_ZCLIENXX-NCLIE.
ENDIF.
* Inicializamos el contenido del campo.
CLEAR I_ZCLIENXX-NCLIE.
* Si nivel 1, mostrar detalle de factura
WHEN 1.
* Comprobamos si es una línea válida, nfact informado
IF ( I_ZFACTUXX-NFACT IS INITIAL ).
MESSAGE E000(38) WITH TEXT-013.
* Si se ha seleccionado una línea válida
ELSE.
* Mostramos el detalle de factura.
PERFORM MOSTRAR_DETALLE_FACTURA USING I_ZFACTUXX-NFACT.
* Inicializamos el contenido del campo.
CLEAR I_ZFACTUXX-NFACT.
ENDIF.
ENDCASE.

*****
* Control de la pantalla de selección. *
*****
* Valores posibles para el rango S_Nombre valor desde.
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-LOW.
* Llamada a la rutina que muestra los valores existentes
PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-LOW.

* Valores posibles para el rango S_Nombre valor hasta
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-HIGH.
* Llamada a la rutina que muestra los valores existentes
PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-HIGH.

*****
* Rutinas adicionales. *
*****

*&-----*
*& Form VALORES_POSIBLES_NOMBRE
*&-----*
* Muestra una ventana con los distintos nombres existentes en *
* la tabla maestro de clientes ZCLIENXX. Dando la posibilidad de *
* seleccionar uno de ellos. Devuelve en el parámetro PS_NOMBR el *
* nombre seleccionado o vacío si no se realiza selección. *
*-----*
* -->PS_NOMBR Nombre seleccionado *
*-----*
FORM VALORES_POSIBLES_NOMBRE USING PS_NOMBR LIKE ZCLIENXX-
NOMBR.

```

* Variables locales *

* Tabla interna para almacenar los campos y atributos a visualizar.

```
DATA : BEGIN OF I_FIELDS OCCURS 1.
      INCLUDE STRUCTURE HELP_VALUE.
DATA: END OF I_FIELDS.
```

* Tabla interna para almacenar el valor de campos a mostrar.

```
DATA : BEGIN OF I_VALUES OCCURS 0,
      VALOR(20) TYPE C,
      END OF I_VALUES.
```

* Proceso *

* Insertamos en la tabla I_FIELDS los atributos del campo a mostrar

```
CLEAR I_FIELDS.
I_FIELDS-TABNAME = 'ZCLIENXX'.    " Nombre de la tabla
I_FIELDS-FIELDNAME = 'NOMBR'.    " Nombre del campo
I_FIELDS-SELECTFLAG = 'X'.      " Valor seleccionable
APPEND I_FIELDS.
```

* Seleccionamos los distintos nombres existentes en la tabla ZCLIENXX

* y los almacenamos en la tabla de valores

```
SELECT DISTINCT NOMBR INTO ZCLIENXX-NOMBR
FROM ZCLIENXX
WHERE BUKRS = P_BUKRS.
I_VALUES-VALOR = ZCLIENXX-NOMBR.
APPEND I_VALUES.
ENDSELECT.
```

* Llamamos a la función predefinida que muestra los datos

```
CALL FUNCTION 'HELP_VALUES_GET_WITH_TABLE'
```

```
* EXPORTING
*   CUCOL           = 0
*   CUROW           = 0
*   DISPLAY         = ''
*   FIELDNAME       = ''
*   TABNAME         = ''
*   NO_MARKING_OF_CHECKVALUE = ''
*   TITLE_IN_VALUES_LIST = ''
*   TITEL           = ''
*   SHOW_ALL_VALUES_AT_FIRST_TIME = ''
*   NO_CONVERSION   = ''
IMPORTING
  SELECT_VALUE      = PS_NOMBR
TABLES
  FIELDS            = I_FIELDS
```

```

        VALUETAB          = I_VALUES
    EXCEPTIONS
        FIELD_NOT_IN_DDIC      = 1
        MORE_THAN_ONE_SELECTFIELD = 2
        NO_SELECTFIELD        = 3
        OTHERS                 = 4.
ENDFORM.          " VALORES_POSIBLES_NOMBRE
*&-----*
*&   Form  MOSTRAR_FACTURAS_CLIENTE
*&-----*
*   Selecciona las facturas correspondientes al cliente recibido *
* como parámetro y las muestra en pantalla. *
* Si no se seleccionan facturas, se muestra el mensaje correspondiente
*-----*
*   -->PE_CLIENX  -- N° de cliente. *
*-----*
FORM  MOSTRAR_FACTURAS_CLIENTE  USING  VALUE(PE_NCLIE)  LIKE
ZCLIENXX-NCLIE.

*****
* Proceso *
*****

* Seleccionamos las facturas del cliente para la sociedad indicada
SELECT *
FROM ZFACTUXX INTO TABLE I_ZFACTUXX
WHERE BUKRS = P_BUKRS
AND  NCLIE = PE_NCLIE.
* Si se han seleccionado facturas
IF ( SY-SUBRC = 0 ).
* Activamos color de detalle
FORMAT COLOR COL_NORMAL.

* Para cada una de las facturas seleccionadas
LOOP AT I_ZFACTUXX.
* Escribimos los datos de las facturas.
WRITE: AT /C_POS_NFACT '|' NO-GAP, I_ZFACTUXX-NFACT.
* Almacenamos el valor del n° de factura.
HIDE I_ZFACTUXX-NFACT.

WRITE: AT C_POS_FEFAC '|' NO-GAP, I_ZFACTUXX-FECHA,
        AT C_POS_MESFA '|' NO-GAP, I_ZFACTUXX-MESFA,
        AT C_POS_IMPNT '|' NO-GAP, I_ZFACTUXX-IMPNT CURRENCY
        I_ZFACTUXX-MONED,
        AT C_POS_MONED '|' NO-GAP, I_ZFACTUXX-MONED,
        AT C_POS_FIFAC '|' NO-GAP.
ENDLOOP.
* Escribimos línea final

```

```

      ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
* Si no se han seleccionado
      ELSE.
* Mostramos mensaje de información con
* 'No existen facturas para el cliente:'xxx 'para la sociedad:' xxx
      MESSAGE I000(38) WITH TEXT-015 PE_NCLIE TEXT-016 P_BUKRS.
      ENDIF.
ENDFORM.          " MOSTRAR_FACTURAS_CLIENTE
*&-----*
*&   Form MOSTRAR_DETALLE_FACTURA
*&-----*
*   Muestra el detalle de una factura, mostrando el cliente      *
* al que pertenece y calculando el IVA correspondiente a la misma *
* su poniendo un tipo de IVA fijo del 16 %.                      *
*-----*
*   --> PE_NFACT  Código de factura                               *
*-----*
FORM  MOSTRAR_DETALLE_FACTURA  USING  VALUE(PE_NFACT)  LIKE
ZFACTUXX-NFACT.

*****
* Definición de variables locales                                *
*****

DATA: D_NOMBRE_CLIENTE(70) TYPE C,      " Nombre completo
      D_IMPORT_IVA  LIKE ZFACTUXX-IMPNT, " Importe del IVA
      D_IMPORT_TOTAL LIKE ZFACTUXX-IMPNT. " Importe total

*****
* Proceso                                                       *
*****

* Obtenemos los datos de la factura para ello leemos de la tabla interna
      READ TABLE I_ZFACTUXX WITH KEY NFACT = PE_NFACT.
* Comprobamos que existe
      CHECK ( SY-SUBRC = 0 ).
* Obtenemos los datos necesarios del cliente
      SELECT SINGLE NOMBR APEL1 APEL2
      FROM ZCLIENXX INTO (ZCLIENXX-NOMBR, ZCLIENXX-APEL1, ZCLIENXX-APEL2)
      WHERE BUKRS = P_BUKRS
      AND  NCLIE = I_ZFACTUXX-NCLIE.
* Comprobamos que existe.
      CHECK ( SY-SUBRC = 0 ).
* Escribimos los datos en la forma adecuada
      ULINE AT /C_POS_D_INICI(C_POS_D_FINAL).
* Nº de factura.
      WRITE: AT /C_POS_D_INICI '|',
            AT C_POS_D_NFACT TEXT-022, I_ZFACTUXX-NFACT,

```

AT C_POS_D_FINAL '|' NO-GAP.

* **Formamos el nombre completo del cliente del cliente**

CONCATENATE ZCLIENXX-NOMBR ZCLIENXX-APEL1 ZCLIENXX-APEL2
INTO D_NOMBRE_CLIENTE SEPARATED BY SPACE.

* **Nombre del cliente**

WRITE: AT /C_POS_D_INICI '|',TEXT-023, D_NOMBRE_CLIENTE,
AT C_POS_D_FINAL '|' NO-GAP.

* **Fecha de factura**

WRITE: AT /C_POS_D_INICI '|', TEXT-024, I_ZFACTUXX-FECHA,
AT C_POS_D_FINAL '|' NO-GAP.

* **Mes de la factura.**

WRITE: AT /C_POS_D_INICI '|', TEXT-025, I_ZFACTUXX-MESFA,
AT C_POS_D_FINAL '|' NO-GAP.

* **Moneda de la factura.**

WRITE: AT /C_POS_D_INICI '|', TEXT-026, I_ZFACTUXX-MONED,
AT C_POS_D_FINAL '|' NO-GAP.

* **Importe neto:**

WRITE: AT /C_POS_D_INICI '|',
AT C_POS_D_IMPOR TEXT-027,
I_ZFACTUXX-IMPNT CURRENCY I_ZFACTUXX-MONED,
AT C_POS_D_FINAL '|' NO-GAP.

* **Calculamos el importe de IVA. (16 %).**

$D_IMPOR_IVA = (I_ZFACTUXX-IMPNT * 16) / 100.$

WRITE: AT /C_POS_D_INICI '|',
AT C_POS_D_IMPOR TEXT-028,
D_IMPOR_IVA CURRENCY I_ZFACTUXX-MONED,
AT C_POS_D_FINAL '|' NO-GAP.

* **Calculamos el importe total**

$D_IMPOR_TOTAL = I_ZFACTUXX-IMPNT + D_IMPOR_IVA.$

WRITE: AT /C_POS_D_INICI '|',
AT C_POS_D_IMPOR TEXT-029,
D_IMPOR_TOTAL CURRENCY I_ZFACTUXX-MONED,
AT C_POS_D_FINAL '|' NO-GAP.

* **Mostramos línea final**

ULINE AT /C_POS_D_INICI(C_POS_D_FINAL).

ENDFORM. " MOSTRAR_DETALLE_FACTURA

4.3.6 Ejemplo de listado interactivo II.

Vamos a ver otro ejemplo de listado interactivo, es este caso va a consistir en mostrar un listado con los clientes existentes y poder marcar y desmarcar algunos de ellos para su posterior tratamiento.

Para ello vamos a realizar una copia del programa anterior 'ZREPO3XX' a la que llamaremos 'ZREPO4XX' y sobre este realizaremos las siguientes modificaciones:

En la tabla interna I_ZCLIENXX, será necesario añadir un nuevo campo (CHAR 1) para poder ser usado como campo para marcar y desmarcar. Para ello definiremos la tabla del siguiente modo:

* **Tabla interna de datos a listar**

```
DATA: BEGIN OF I_ZCLIENXX OCCURS 0.  
DATA MARCA(1) TYPE C.           " Campo para marcar  
      INCLUDE STRUCTURE ZCLIENXX.  
DATA: END OF I_ZCLIENXX.
```

Al incluir un nuevo campo en la tabla tendremos que tenerlo en cuenta en la instrucción SELECT (de acceso a la tabla ZCLIENXX) ya que ahora la instrucción habrá que realizar el siguiente cambio.

```
FROM ZCLIENXX INTO TABLE I_ZCLIENXX.
```

Por

```
FROM ZCLIENXX INTO CORRESPONDING FIELDS OF TABLE I_ZCLIENXX.
```

Si no hiciéramos esto, estaríamos recuperando la información pero el valor se volcaría de forma errónea sobre los campos.

También modificaremos el evento TOP-OF-PAGE para ajustar la cabecera al nuevo campo. (Deberemos cambiar también el valor de las constantes para llevar el listado dos posiciones a la derecha para abrir hueco al campo Marca).

Será necesario además escribir el campo en el listado en forma de casilla de selección y de entrada para que pueda ser marcado y desmarcado. Para ello modificamos el evento END-OF-SELECTION al que añadiremos:

- * **Escribimos el campo MARCA en forma de casilla de selección**
 - * **y con los atributos de campo de entrada.**
- ```
WRITE: AT /C_POS_INICI '|' NO-GAP,
 I_ZCLIENXX-MARCA AS CHECKBOX INPUT ON.
```

Con la inclusión de estos cambios el listado tendrá el siguiente aspecto:

| N.Cliente | Nombre   | Primer Apellido | Segundo Apellido | F.Nacim. |
|-----------|----------|-----------------|------------------|----------|
| 000000001 | RAFAEL   | PEREZ           | LOPEZ            | 01.12.19 |
| 000000002 | RAFAEL   | MARTINEZ        | SAN JUAN         | 01.01.19 |
| 000000003 | JUAN     | MARTINEZ        | SANCHEZ          | 01.01.19 |
| 000000004 | JOSE     | DE PABLO        | SANCHEZ          | 01.01.19 |
| 000000005 | ADOLFO   | QUILEZ          | GOMEZ            | 01.02.19 |
| 000000006 | JORGE    | RICOTE          | GONZALEZ         | 20.05.19 |
| 000000007 | MERCEDES | CALERO          | SANCHEZ          | 01.07.19 |

El siguiente paso será dar la posibilidad de mostrar una ventana con las facturas con los clientes seleccionados, mediante la opción de un botón en la barra de pulsadores.

Para crear opciones de usuario, bien como botones en la barra de pulsadores, bien como opciones de menú en la barra de menús, es necesario definir un STATUS.

Existen cuatro tipos de STATUS dependiendo de su utilización:

- Dynpro
- Ventana de diálogo
- Lista
- Lista en Ventana de dialogo

Los dos primeros se utilizan en pantallas (Programación de dialogo que se verá en capítulos posteriores).

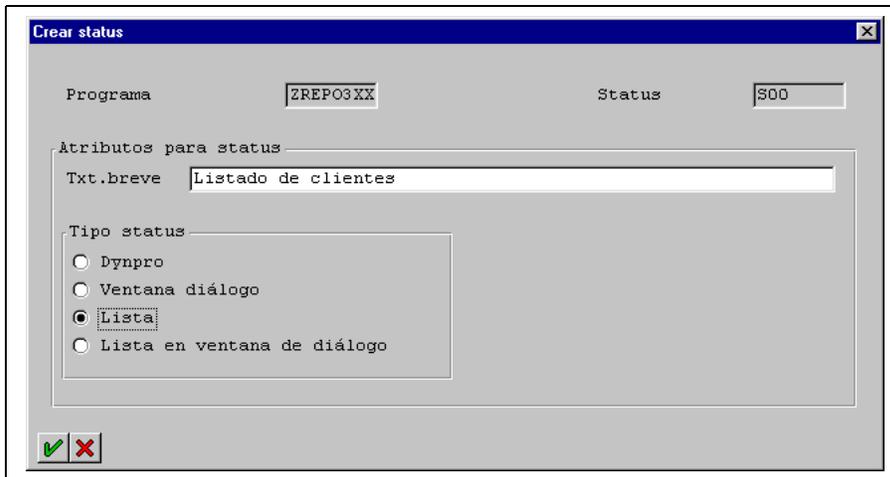
Los dos siguientes se utilizan en la programación de listados interactivos. El tipo lista es el que se utiliza normalmente en los listados a pantalla completa, quedando la lista en ventana de diálogo para los listados en ventanas. No obstante podemos utilizar un tipo Lista en una ventana y viceversa.

Vamos a crear un STATUS tipo lista al que llamaremos 'S00' para ello escribimos la instrucción para activar el STATUS, en nuestro caso, la situaremos al comienzo del evento END-OF-SELECTION

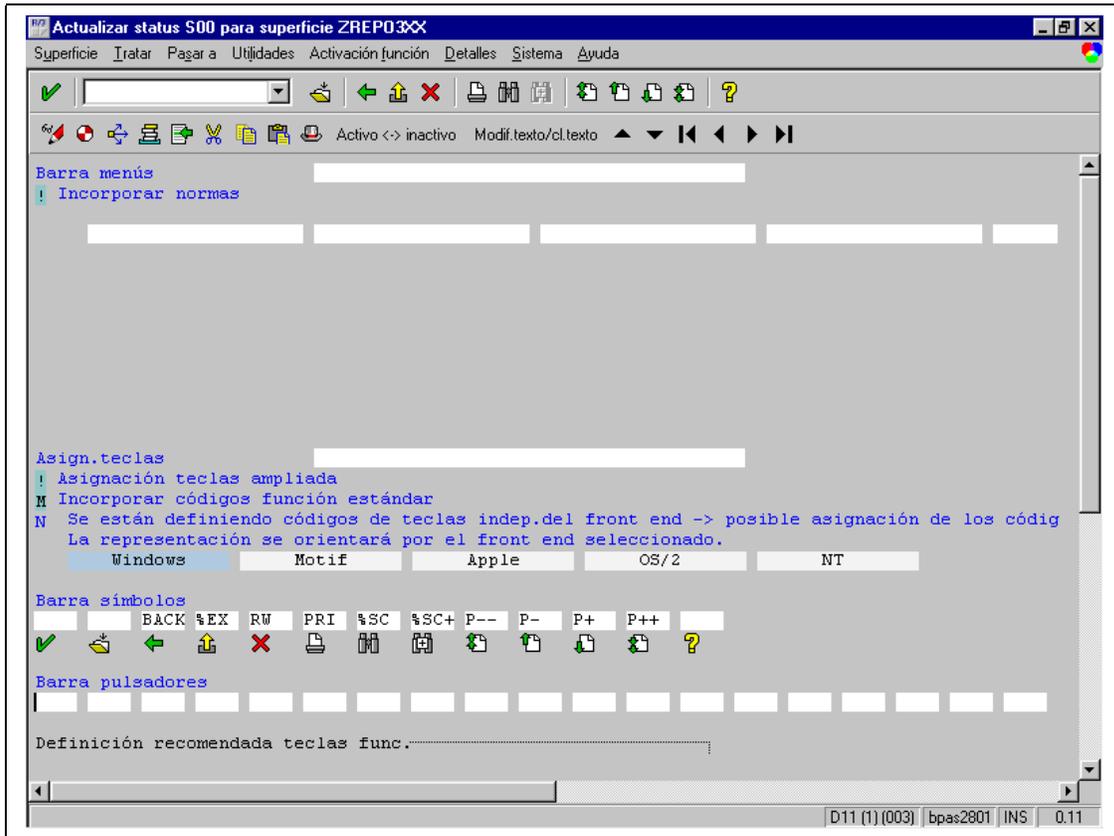
SET PF-STATUS 'S00'.

Esta instrucción provocará que se llame al STATUS S00 en lugar de al STATUS por defecto que se llamaba hasta ahora.

Usamos la utilidad del editor de navegar al objeto haciendo Doble-Click sobre él , nos saldrá una ventana indicando que el Status no existe, pero nos da la posibilidad de crearlo, nos aparecerá la siguiente ventana donde deberemos indicar los atributos del STATUS, pondremos '*Listado clientes*' en texto breve y seleccionaremos el tipo '*Lista*'.

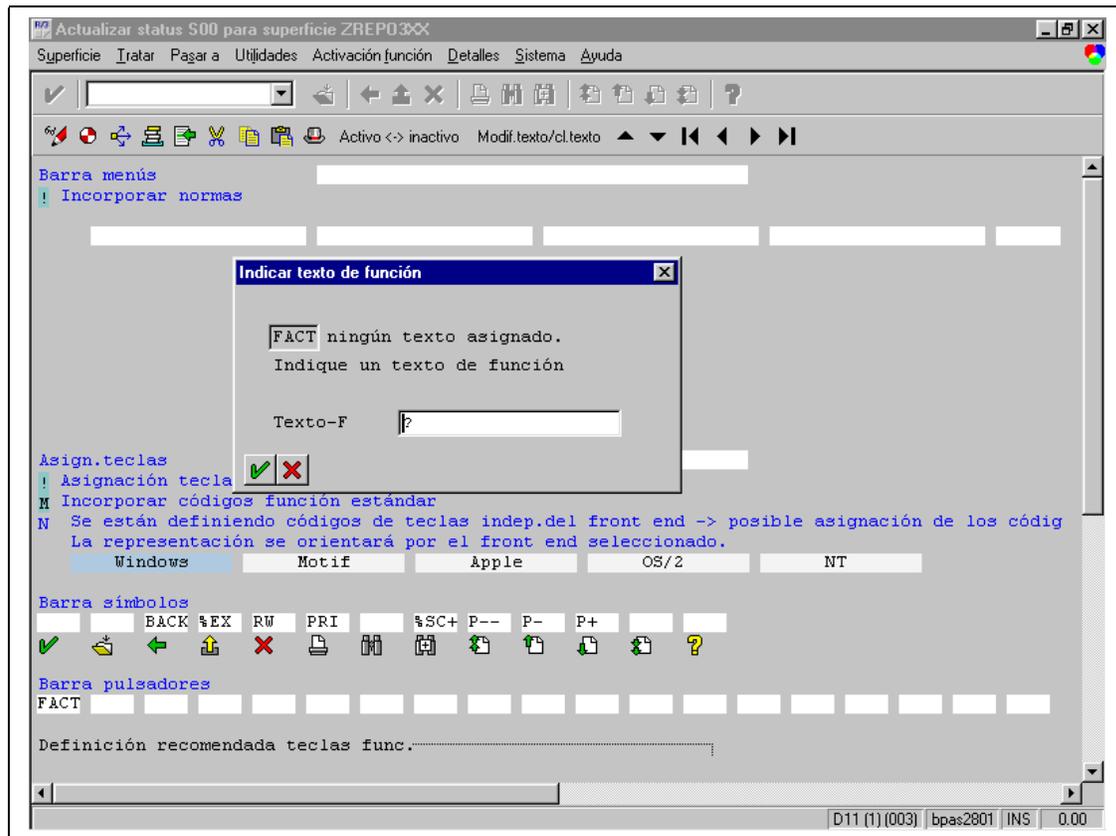


Pulsamos continuar, llegaremos a la siguiente pantalla de para la definición de STATUS:



Por defecto, aparecen todas las opciones existentes en los listados (En los ejemplos anteriores teníamos ya estas opciones sin crear nuestro propio STATUS), podemos eliminar estas opciones, borrando el código de comando correspondiente de la barra de símbolos. (En *nuestro caso no vamos a eliminar ninguna*).

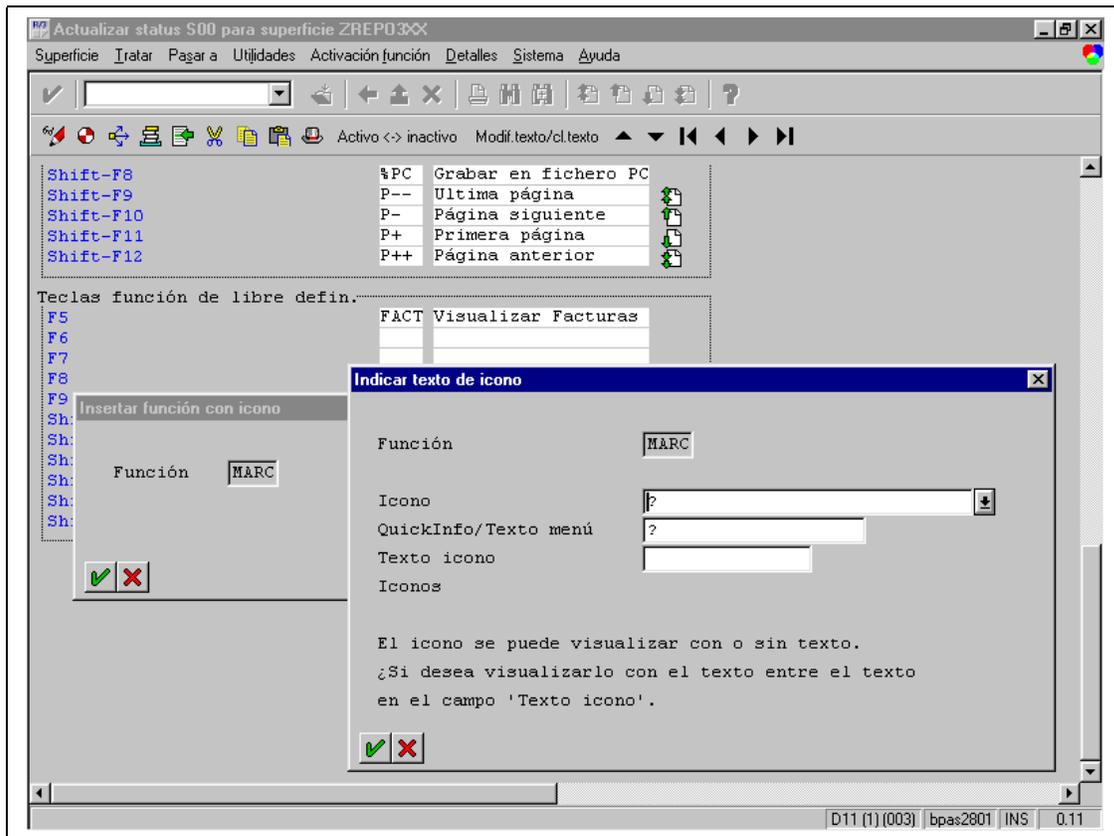
Creamos nuestro botón facturas, para ello, escribimos el código de comando que vamos a asociar al botón en el primer hueco disponible de la barra de pulsadores, pondremos 'FACT' al pulsar ENTER nos saldrá una ventana para elegir la tecla de función asociada al comando, seleccionaremos 'F5', a continuación nos saldrá la siguiente ventana:



Donde pondremos el texto que mostrará el botón, 'Visualizar Facturas' en nuestro caso. Tras aceptar tendremos nuestro primer botón.

Aprovechamos para incluir dos nuevos botones, un botón que nos servirá para marcar a todos los clientes del listado y otro que permita desmarcarlos. Estos botones van a tener la particularidad de que van a tener asociado un Icono.

Vamos a crear el botón 'MARCAR' al que le asociaremos el ICONO (  ). Para ello, nos situamos en la primera tecla de función de libre definición, que este libre (las veremos haciendo un avance de página) en nuestro caso ('F6'), seleccionamos la opción del menú 'Tratar→Insertar→Función con Icono (Ctrl. + F10), nos aparecerá una ventana donde pondremos el código de comando asociado 'MARC' en nuestro caso, nos llevará a la siguiente ventana:



**ICONO:** Es el nombre del icono, que se mostrará. (En nuestro caso seleccionaremos el icono 'ICON\_SELECT\_ALL' ()).

**QuickInfo/ Texto menú:** Es el texto de información que se mostrará en la ventana de ayuda que sale al situarse sobre un botón. (En nuestro caso pondremos 'Marcar todos'.)

**Texto icono:** Corresponde al texto que tendrá el botón además del icono. (En nuestro caso lo dejamos en blanco).

Tras pulsar aceptar aparecerá la opción en la línea correspondiente, ('F6' en nuestro caso).

Repetimos el proceso para el botón 'DESMARCAR' asociándole a la tecla de función 'F7', el comando 'DESM', el ICONO 'ICON\_DESELECT\_ALL' () y el texto de ayuda 'Desmarcar todos'.

Una vez definidos ambos comandos, escribiremos, MARC y DESM en la barra de pulsadores. Junto a ellos añadiremos el comando predefinido 'PICK' correspondiente a la selección de líneas (Este es el que hasta ahora se activa al hacer Click y Doble-Click) de esta manera queda más claro en que listados está disponible la opción de selección.

A continuación grabaremos y activaremos el STATUS.

Será necesario definirse un nuevo STATUS con las siguientes propiedades:

**S01:** Será de tipo lista en ventana de diálogo, será el que activemos cuando llamemos a la ventana donde mostrar las facturas de los clientes.

Tendrá las opciones que se crean por defecto más la opción 'PICK' que añadiremos a la barra de pulsadores.

Para el listado de detalle de factura, no será necesario definirse ningún STATUS ya que utilizaremos el STATUS estándar, activándolo con 'SET PF-STATUS SPACE'.

De vuelta al código del programa ya tendremos un nuevo STATUS con las opciones disponibles. Ahora falta realizar el tratamiento de dichas opciones. Este tratamiento se realiza en el evento AT USER-COMMAND, en el que hay que controlar:

- Visualizar facturas:  
Para visualizar las facturas de los clientes seleccionados, será necesario, hacer dos cosas.  
En primer lugar, leer las marcas del listado, es decir, ver que clientes están marcados en pantalla, actualizaremos el campo marca en la tabla interna I\_ZCLIENES. Esta tarea se realiza dentro del procedimiento 'LEER\_MARCAS\_LISTADO'.  
Por otra parte será necesario mostrar las facturas de aquellos clientes seleccionados, es decir, de aquellos clientes que tengan el campo marca activado. Esto se hace dentro del procedimiento 'MOSTRAR\_FACTURAS\_CLIENTES' en el que se activa el STATUS 'S01' correspondiente al listado de facturas y se activa el listado en forma de ventana utilizando la instrucción **WINDOW STARTING AT... ENDING AT...** En la llamada al STATUS, eliminamos la opción de selección para impedir que se muestre el detalle de la factura. Posteriormente se llama al procedimiento, ya existente, 'MOSTRAR\_FACTURAS\_CLIENTE' para mostrar las facturas de cada cliente seleccionado.
- Marcar todos los clientes.  
Para marcar y visualizar en el listado las marcas, es necesario:  
Por una parte modificar el campo MARCA de todos los clientes de la tabla interna I\_ZCLIENXX para ponerles a 'X', utilizamos para ello el procedimiento 'MARCAR\_DESMARCAR' al que le pasamos como parámetro el valor 'X'.  
Una vez modificado este campo será necesario 'refrescar' el listado, para ello, tendremos que escribir de nuevo el listado con los clientes, permaneciendo en el mismo nivel de lista, esto lo controlamos descontando uno a la variable SY-LSIND.
- Desmarcar todos los clientes.  
El procedimiento para desmarcar los clientes es similar, a diferencia de que actualizamos el campo marca con el valor SPACE.

Cambiamos el modo de controlar los niveles de lista, utilizaremos la variable SY-PFKEY que nos indica el STATUS actual, en función de este, mostraremos una cabecera u otra un listado u otro, en los eventos AT LINE-SELECTION, AT TOP-PAGE DURING LINE-SELECTION.

El listado completo del programa será algo así:

```

* PROGRAMA: ZREPO4XX. *
* DESCRIPCION: Muestra un listado de los clientes existentes en el *
* que se detalla en nº de cliente junto con el nombre y *
* apellidos . Así mismo se podrán visualizar las facturas *
* de un cliente/s y también el detalle de las mismas. *
* AUTOR : Nombre y Apellidos FECHA: 13/08/2001 *

* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *

REPORT ZREPO4XX NO STANDARD PAGE HEADING LINE-SIZE 120 LINE-COUNT
80.

* Tablas del diccionario de datos *

TABLES: ZCLIENXX, " Maestro de clientes.
ZFACTUXX. " Facturas clientes

* Definición de constantes *

CONSTANTS:
* Constantes posiciones clientes
C_POS_INICI(3) TYPE N VALUE 1, " Posición de inicio
C_POS_NCLIE(3) TYPE N VALUE 3, " Posición nº cliente
C_POS_NOMBR(3) TYPE N VALUE 14, " Posición nombre
C_POS_APEL1(3) TYPE N VALUE 35, " Posición apellido1
C_POS_APEL2(3) TYPE N VALUE 61, " Posición apellido2
C_POS_FNACI(3) TYPE N VALUE 87, " Posición F. Nacimiento
C_POS_FINAL(3) TYPE N VALUE 98, " Posición final
C_ANCHO_TOTAL(3) TYPE N VALUE 98, " Ancho total del informe,
C_POS_TITUL(3) TYPE N VALUE 1, " Posición título,
C_POS_USUAR(3) TYPE N VALUE 37, " Posición usuario
C_POS_FECHA(3) TYPE N VALUE 60, " Posición fecha
C_POS_PAGNO(3) TYPE N VALUE 85, " Posición nº de página
* Constantes posiciones facturas.
C_POS_NFACT(3) TYPE N VALUE 1, " Posición Nº de factura
C_POS_FEFAC(3) TYPE N VALUE 12, " Posición Fecha
C_POS_MESFA(3) TYPE N VALUE 23, " Posición Mes
C_POS_IMPNT(3) TYPE N VALUE 34, " Posición Importe
C_POS_MONED(3) TYPE N VALUE 51, " Posición moneda
C_POS_FIFAC(3) TYPE N VALUE 57, " Posición final factura.
C_ANCHO_FAC(3) TYPE N VALUE 57, " Ancho detalle factura.
```

\* Constantes de posiciones de detalle de factura

C\_POS\_D\_INICI(3) TYPE N VALUE 1, " Posición inicial  
C\_POS\_D\_FINAL(3) TYPE N VALUE 80, " Posición final  
C\_POS\_D\_NFACT(3) TYPE N VALUE 20, " Posición nfactura  
C\_POS\_D\_IMPORT(3) TYPE N VALUE 20. " Posición importes

\*\*\*\*\*

\* Tablas internas

\*

\*\*\*\*\*

\* Tabla interna de datos a listar de clientes

DATA: BEGIN OF I\_ZCLIENXX OCCURS 0.  
DATA MARCA(1) TYPE C. " Campo para marcar  
INCLUDE STRUCTURE ZCLIENXX.  
DATA: END OF I\_ZCLIENXX.

\* Tabla interna con datos de facturas

DATA: BEGIN OF I\_ZFACTUXX OCCURS 0.  
INCLUDE STRUCTURE ZFACTUXX.  
DATA: END OF I\_ZFACTUXX.

\*\*\*\*\*

\* Pantalla de selección

\*

\*\*\*\*\*

\* Parámetros

SELECTION-SCREEN BEGIN OF BLOCK BLOQ1 WITH FRAME TITLE TEXT-001.  
PARAMETERS P\_BUKRS LIKE ZCLIENXX-BUKRS OBLIGATORY  
DEFAULT '0001'. " Sociedad  
SELECTION-SCREEN END OF BLOCK BLOQ1.

\* Rangos de selección

SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE TEXT-002.  
SELECT-OPTIONS:

\* Añadimos el match-code para el código de cliente

S\_NCLIE FOR ZCLIENXX-NCLIE MATCHCODE OBJECT ZCXX, " N° de cliente  
S\_NOMBR FOR ZCLIENXX-NOMBR, " Nombre cliente  
S\_FNACI FOR ZCLIENXX-FNACI. " Fecha de nacimiento  
SELECTION-SCREEN END OF BLOCK BLOQ2.

\*\*\*\*\*

\* Comienzo de selección

\*

\*\*\*\*\*

START-OF-SELECTION.

\* Inicializamos la tabla interna.

REFRESH I\_ZCLIENXX.  
CLEAR I\_ZCLIENXX.

\* Seleccionamos los datos de los clientes que cumplen criterios

SELECT \*  
FROM ZCLIENXX INTO CORRESPONDING FIELDS OF TABLE I\_ZCLIENXX

```
WHERE BUKRS = P_BUKRS
AND NCLIE IN S_NCLIE
AND NOMBR IN S_NOMBR
AND FNACI IN S_FNACI.
* Comprobamos si no hay datos seleccionados en cuyo caso,
* escribiremos un texto que lo indique.
IF NOT (SY-SUBRC = 0).
 WRITE / TEXT-003. " No hay datos para la selección indicada
ENDIF.
```

```

* Final de la selección *

```

```
END-OF-SELECTION.
* Activamos el status de listado de clientes
SET PF-STATUS 'S00'.
* Llamamos al procedimiento que muestra el listado de clientes
PERFORM MOSTRAR_CLIENTES.
```

```

* Cabecera de página *

```

```
TOP-OF-PAGE.
* Llamamos al proce. que escribe la cabecera del listado de clientes
PERFORM MOSTRAR_CABECERA_CLIENTES.
```

```

* Cabecera de página durante la selección *

```

```
TOP-OF-PAGE DURING LINE-SELECTION.
* Comprobamos el STATUS activo en función del cual mostraremos la
* cabecera correspondiente
CASE SY-PFKEY.
* Si status S00, activa cabecera clientes
WHEN 'S00'.
* Llamamos al proce. que escribe cabecera de clientes
PERFORM MOSTRAR_CABECERA_CLIENTES.
* Si status S01, activamos cabecera facturas
WHEN 'S01'.
* Escribimos una línea horizontal
ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
* Activamos color de cabecera
FORMAT COLOR COL_HEADING.
* Escribimos el detalle de la cabecera de factura.
WRITE: AT /C_POS_NFACT '|' NO-GAP, TEXT-017,
 AT C_POS_FEFAC '|' NO-GAP, TEXT-018,
 AT C_POS_MESFA '|' NO-GAP, TEXT-019,
 AT C_POS_IMPNT '|' NO-GAP, TEXT-020,
```

```
AT C_POS_MONED '|' NO-GAP, TEXT-021,
AT C_POS_FIFAC '|' NO-GAP.
* Escribimos una línea horizontal
 ULINE AT /C_POS_NFACT(C_ANCHO_FAC).

ENDCASE.

* Evento de selección de línea *

AT LINE-SELECTION.
* En función del STATUS existente
CASE SY-PFKEY.
* Si Listado de clientes 'S00'.
WHEN 'S00'.
* Comprobamos si es una línea válida,
* es decir el campo de código de cliente está informado
 IF (I_ZCLIENXX-NCLIE IS INITIAL).
 MESSAGE E000(38) WITH TEXT-013.
* Si se ha seleccionado un línea válida.
 ELSE.
* Activamos el STATUS del listado de facturas
 SET PF-STATUS 'S01'.
* Llamamos al procedimiento para mostrar las facturas del cliente.
 PERFORM MOSTRAR_FACTURAS_CLIENTE USING I_ZCLIENXX-NCLIE.
 ENDIF.
* Inicializamos el contenido del campo.
 CLEAR I_ZCLIENXX-NCLIE.
* Si listado de facturas 'S01'.
WHEN 'S01'.
* Comprobamos si es una línea válida, nfact informado
 IF (I_ZFACTUXX-NFACT IS INITIAL).
 MESSAGE E000(38) WITH TEXT-013.
* Si se ha seleccionado una línea válida
 ELSE.
* Activamos el STATUS por defecto.
 SET PF-STATUS SPACE.
* Mostramos el detalle de factura.
 PERFORM MOSTRAR_DETALLE_FACTURA USING I_ZFACTUXX-NFACT.
* Inicializamos el contenido del campo.
 CLEAR I_ZFACTUXX-NFACT.
 ENDIF.
ENDCASE.

* Evento de comandos de usuario *

AT USER-COMMAND.
```

```
* En función del comando seleccionado
CASE SY-UCOMM.
* Si opción facturas
 WHEN 'FACT'.
* Leemos los clientes seleccionados
 PERFORM LEER_MARCAS_LISTADO.
* Mostramos las facturas de los clientes
 PERFORM MOSTRAR_FACTURAS_CLIENTES.
* Si opción de marcar todos los clientes
 WHEN 'MARC'.
* Marcamos a todos los cliente desmarcados
 PERFORM MARCAR_DESMARCAR USING 'X'.
* Mantenemos el mismo nivel de lista actual
 SY-LSIND = SY-LSIND - 1.
* Reescribimos el listado en los nuevos valores actualizados
 PERFORM MOSTRAR_CLIENTES.
* Si opción de desmarcar todos los clientes.
 WHEN 'DESM'.
* Desmarcamos a todos los clientes marcados
 PERFORM MARCAR_DESMARCAR USING SPACE.
* Mantenemos el mismo nivel de lista actual
 SY-LSIND = SY-LSIND - 1.
* Reescribimos el listado en los nuevos valores actualizados
 PERFORM MOSTRAR_CLIENTES.
ENDCASE.

* Control de la pantalla de selección.

* Valores posibles para el rango S_Nombre valor desde.
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-LOW.
* Llamada a la rutina que muestra los valores existentes
 PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-LOW.

* Valores posibles para el rango S_Nombre valor hasta
AT SELECTION-SCREEN ON VALUE-REQUEST FOR S_NOMBR-HIGH.
* Llamada a la rutina que muestra los valores existentes
 PERFORM VALORES_POSIBLES_NOMBRE USING S_NOMBR-HIGH.

* Rutinas adicionales.

&-----
*& Form VALORES_POSIBLES_NOMBRE
&-----
* Muestra una ventana con los distintos nombres existentes en
* la tabla maestro de clientes ZCLIENXX. Dando la posibilidad de
* seleccionar uno de ellos. Devuelve en el parámetro PS_NOMBR el
```

```

* nombre seleccionado o vacio si no se realiza selección.

* -->PS_NOMBR Nombre seleccionado

FORM VALORES_POSIBLES_NOMBRE USING PS_NOMBR LIKE ZCLIENXX-
NOMBR.

* Variables locales

* Tabla interna para almacenar los campos y atributos a visualizar.
DATA : BEGIN OF I_FIELDS OCCURS 1.
 INCLUDE STRUCTURE HELP_VALUE.
DATA: END OF I_FIELDS.
* Tabla interna para almacenar el valor de campos a mostrar.
DATA : BEGIN OF I_VALUES OCCURS 0,
 VALOR(20) TYPE C,
 END OF I_VALUES.

* Proceso

* Insertamos en la tabla I_FIELDS los atributos del campo a mostrar
CLEAR I_FIELDS.
I_FIELDS-TABNAME = 'ZCLIENXX'. " Nombre de la tabla
I_FIELDS-FIELDNAME = 'NOMBR'. " Nombre del campo
I_FIELDS-SELECTFLAG = 'X'. " Valor seleccionable
APPEND I_FIELDS.
* Seleccionamos los distintos nombres existentes en la tabla ZCLIENXX
* y los almacenamos en la tabla de valores
SELECT DISTINCT NOMBR INTO ZCLIENXX-NOMBR
FROM ZCLIENXX
WHERE BUKRS = P_BUKRS.
I_VALUES-VALOR = ZCLIENXX-NOMBR.
APPEND I_VALUES.
ENDSELECT.

* Llamamos a la función predefinida que muestra los datos
CALL FUNCTION 'HELP_VALUES_GET_WITH_TABLE'
* EXPORTING
* CUCOL = 0
* CUROW = 0
* DISPLAY = ''
* FIELDNAME = ''
* TABNAME = ''
* NO_MARKING_OF_CHECKVALUE = ''
* TITLE_IN_VALUES_LIST = ''
* TITEL = ''

```

```

* SHOW_ALL_VALUES_AT_FIRST_TIME = ''
* NO_CONVERSION = ''
IMPORTING
 SELECT_VALUE = PS_NOMBR
TABLES
 FIELDS = I_FIELDS
 VALUETAB = I_VALUES
EXCEPTIONS
 FIELD_NOT_IN_DDIC = 1
 MORE_THAN_ONE_SELECTFIELD = 2
 NO_SELECTFIELD = 3
 OTHERS = 4.
ENDFORM. " VALORES_POSIBLES_NOMBRE
&-----
*& Form MOSTRAR_FACTURAS_CLIENTE
&-----
* Selecciona las facturas correspondientes al cliente recibido *
* como parámetro y las muestra en pantalla. *
* Si no se seleccionan facturas, se muestra el mensaje correspondiente

* -->PE_CLIENX -- N° de cliente. *

FORM MOSTRAR_FACTURAS_CLIENTE USING VALUE(PE_NCLIE) LIKE
ZCLIENXX-NCLIE.

* Proceso *

* Seleccionamos las facturas del cliente para la sociedad indicada
SELECT *
FROM ZFACTUXX INTO TABLE I_ZFACTUXX
WHERE BUKRS = P_BUKRS
AND NCLIE = PE_NCLIE.
* Si se han seleccionado facturas
IF (SY-SUBRC = 0).
* Activamos color de detalle
FORMAT COLOR COL_NORMAL.

* Para cada una de las facturas seleccionadas
LOOP AT I_ZFACTUXX.
* Escribimos los datos de las facturas.
WRITE: AT /C_POS_NFACT '|' NO-GAP, I_ZFACTUXX-NFACT.
* Almacenamos el valor del n° de factura.
HIDE I_ZFACTUXX-NFACT.

WRITE: AT C_POS_FEFAC '|' NO-GAP, I_ZFACTUXX-FECHA,
 AT C_POS_MESFA '|' NO-GAP, I_ZFACTUXX-MESFA,

```

```

 AT S_IMPNT '|' NO-GAP, I_ZFACTUXX-IMPNT
 CURRENCY I_ZFACTUXX-MONED,
 AT C_POS_MONED '|' NO-GAP, I_ZFACTUXX-MONED,
 AT C_POS_FIFAC '|' NO-GAP.
 ENDLOOP.
* Escribimos línea final
 ULINE AT /C_POS_NFACT(C_ANCHO_FAC).
* Si no se han seleccionado
 ELSE.
* Mostramos mensaje de Status con
* 'No existen facturas para el cliente:'xxx 'para la sociedad:' xxx
 MESSAGE S000(38) WITH TEXT-015 PE_NCLIE TEXT-016 P_BUKRS.
 ENDIF.
ENDFORM. " MOSTRAR_FACTURAS_CLIENTE
&-----
*& Form MOSTRAR_DETALLE_FACTURA
&-----
* Muestra el detalle de una factura, mostrando el cliente *
* al que pertenece y calculando el IVA correspondiente a la misma *
* su poniendo un tipo de IVA fijo del 16 %. *

* --> PE_NFACT Código de factura *

FORM MOSTRAR_DETALLE_FACTURA USING VALUE(PE_NFACT) LIKE
ZFACTUXX-NFACT.

* Definición de variables locales *

DATA: D_NOMBRE_CLIENTE(70) TYPE C, " Nombre completo
 D_IMPOR_IVA LIKE ZFACTUXX-IMPNT, " Importe del IVA
 D_IMPOR_TOTAL LIKE ZFACTUXX-IMPNT. " Importe total

* Proceso *

* Obtenemos los datos de la factura para ello leemos de la tabla interna
 READ TABLE I_ZFACTUXX WITH KEY NFACT = PE_NFACT.
* Comprobamos que existe
 CHECK (SY-SUBRC = 0).
* Obtenemos los datos necesarios del cliente
 SELECT SINGLE NOMBR APEL1 APEL2
 FROM ZCLIENXX INTO (ZCLIENXX-NOMBR, ZCLIENXX-APEL1, ZCLIENXX-APEL2)
 WHERE BUKRS = P_BUKRS
 AND NCLIE = I_ZFACTUXX-NCLIE.
* Comprobamos que existe.

```

CHECK ( SY-SUBRC = 0 ).

\* **Escribimos los datos en la forma adecuada**

ULINE AT /C\_POS\_D\_INICI(C\_POS\_D\_FINAL).

\* **Nº de factura.**

WRITE: AT /C\_POS\_D\_INICI '|',  
AT C\_POS\_D\_NFACT TEXT-022, I\_ZFACTUXX-NFACT,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Formamos el nombre completo del cliente del cliente**

CONCATENATE ZCLIENXX-NOMBR ZCLIENXX-APEL1 ZCLIENXX-APEL2  
INTO D\_NOMBRE\_CLIENTE SEPARATED BY SPACE.

\* **Nombre del cliente**

WRITE: AT /C\_POS\_D\_INICI '|',TEXT-023, D\_NOMBRE\_CLIENTE,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Fecha de factura**

WRITE: AT /C\_POS\_D\_INICI '|', TEXT-024, I\_ZFACTUXX-FECHA,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Mes de la factura.**

WRITE: AT /C\_POS\_D\_INICI '|', TEXT-025, I\_ZFACTUXX-MESFA,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Moneda de la factura.**

WRITE: AT /C\_POS\_D\_INICI '|', TEXT-026, I\_ZFACTUXX-MONED,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Importe neto:**

WRITE: AT /C\_POS\_D\_INICI '|',  
AT C\_POS\_D\_IMPOR TEXT-027,  
I\_ZFACTUXX-IMPNT CURRENCY I\_ZFACTUXX-MONED,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Calculamos el importe de IVA. (16 %).**

$D\_IMPOR\_IVA = ( I\_ZFACTUXX-IMPNT * 16 ) / 100.$

WRITE: AT /C\_POS\_D\_INICI '|',  
AT C\_POS\_D\_IMPOR TEXT-028,  
D\_IMPOR\_IVA CURRENCY I\_ZFACTUXX-MONED,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Calculamos el importe total**

$D\_IMPOR\_TOTAL = I\_ZFACTUXX-IMPNT + D\_IMPOR\_IVA.$

WRITE: AT /C\_POS\_D\_INICI '|',  
AT C\_POS\_D\_IMPOR TEXT-029,  
D\_IMPOR\_TOTAL CURRENCY I\_ZFACTUXX-MONED,  
AT C\_POS\_D\_FINAL '|' NO-GAP.

\* **Mostramos línea final**

ULINE AT /C\_POS\_D\_INICI(C\_POS\_D\_FINAL).

ENDFORM. " MOSTRAR\_DETALLE\_FACTURA

\*&-----\*

\*& Form LEER\_MARCAS\_LISTADO

\*&-----\*

\* Actualiza el campo MARCA de la tabla interna I\_ZCLIENXX con el \*

```

* valor actual en el informe.

FORM LEER_MARCAS_LISTADO.

* Definición de variables locales
*

DATA: L_LINEA LIKE SY-INDEX, " N° de línea de pantalla
 L_MARCA(1) TYPE C, " Marca de pantalla
 L_NCLIE LIKE ZCLIENXX-NCLIE. " Cliente de pantalla

* Proceso

* Inicializamos el contador de líneas
 L_LINEA = 0.
* Leemos las líneas todas las líneas de pantalla
DO.
* Incrementamos el contador de líneas
 L_LINEA = L_LINEA + 1.
* Leemos la línea correspondiente
 READ LINE L_LINEA FIELD VALUE I_ZCLIENXX-MARCA INTO L_MARCA
 I_ZCLIENXX-NCLIE INTO L_NCLIE.
* Si no hay error en la lectura
 IF (SY-SUBRC = 0).
* Comprobamos si es una línea de detalle
 CHECK NOT (L_NCLIE IS INITIAL).
* Leemos la entrada de la tabla correspondiente
 READ TABLE I_ZCLIENXX WITH KEY NCLIE = L_NCLIE BINARY SEARCH.
* Si no hay error
 CHECK (SY-SUBRC = 0).
* Comprobamos si el campo marca es distinto al de la tabla
 IF (I_ZCLIENXX-MARCA <> L_MARCA).
* Modificamos la entrada con el valor de pantalla
 I_ZCLIENXX-MARCA = L_MARCA.
 MODIFY I_ZCLIENXX INDEX SY-TABIX.
 ENDIF.
* Si hay error en la lectura significará final de lista, terminamos
 ELSE.
 EXIT.
 ENDIF.
ENDDO.
ENDFORM. " LEER_MARCAS_LISTADO
&-----
*& Form MOSTRAR_FACTURAS_CLIENTES
&-----
* Muestra las facturas de los clientes seleccionados, es decir *
```

```

* de los que tengan el campo MARCA de la tabla interna I_ZCLIENXX *
* un valor 'X'. Para ello llama al procedimiento de mostrar *
* facturas de un cliente.

FORM MOSTRAR_FACTURAS_CLIENTES.

* Proceso *

* Activamos el STATUS para la ventana, eliminando la opción
* de seleccionar para impedir navegar al detalle de factura.
 SET PF-STATUS 'S01' EXCLUDING 'PICK'.
* Abrimos una ventana para mostrar las facturas
 WINDOW STARTING AT 10 5 ENDING AT 68 20.
* Para cada uno de los clientes seleccionados
 LOOP AT I_ZCLIENXX
 WHERE MARCA = 'X'.
* Con cada uno de los clientes llamamos al procedimiento,
* que muestra las facturas de un cliente
 PERFORM MOSTRAR_FACTURAS_CLIENTE USING I_ZCLIENXX-NCLIE.
 ENDLOOP.
ENDFORM. " MOSTRAR_FACTURAS_CLIENTES
&-----
*& Form MARCAR_DESMARCAR
&-----
* Actualizar el campo marca de todos los registros de la tabla *
* interna I_ZCLIENXX con el valor indicado en el parámetro. *

* -->PE_MARCA Valor para marcar ('X' / SPACE) *

FORM MARCAR_DESMARCAR USING VALUE(PE_MARCA) TYPE C.

* Proceso *

* Actualizamos cada uno de los registros de la tabla interna que
* sean distinto al valor al que actualizar.
 LOOP AT I_ZCLIENXX
 WHERE MARCA <> PE_MARCA.
* Actualizamos el valor
 I_ZCLIENXX-MARCA = PE_MARCA.
* Modificamos el registro.
 MODIFY I_ZCLIENXX.
 ENDLOOP.
ENDFORM. " MARCAR_DESMARCAR
&-----
*& Form MOSTRAR_CLIENTES
&-----
* Lista los clientes seleccionados por pantalla, es decir, los *
* clientes seleccionados en la tabla interna I_ZCLIENXX *

```

```

FORM MOSTRAR_CLIENTES.

* Proceso *

* Vamos a escribir el listado con los clientes seleccionados en la
* tabla interna I_ZCLIENXX
LOOP AT I_ZCLIENXX.
* Escribimos el campo MARCA en forma de casilla de selección
* y con los atributos de campo de entrada.
WRITE: AT /C_POS_INICI '|' NO-GAP,
 I_ZCLIENXX-MARCA AS CHECKBOX INPUT ON.
* Escribimos el nº de cliente en color clave, el resto en otro color
FORMAT COLOR COL_KEY.
* Escribimos en modo Click.
WRITE: AT C_POS_NCLIE '|' NO-GAP, I_ZCLIENXX-NCLIE HOTSPOT.
* Almacenamos el valor del campo
HIDE I_ZCLIENXX-NCLIE.
FORMAT COLOR COL_NORMAL.
WRITE: AT C_POS_NOMBR '|' NO-GAP, I_ZCLIENXX-NOMBR,
 AT C_POS_APEL1 '|' NO-GAP, I_ZCLIENXX-APEL1,
 AT C_POS_APEL2 '|' NO-GAP, I_ZCLIENXX-APEL2,
 AT C_POS_FNACI '|' NO-GAP, I_ZCLIENXX-FNACI,
 AT C_POS_FINAL '|' NO-GAP.
ENDLOOP.
* Si se han seleccionado datos, escribimos la línea final.
IF (SY-SUBRC = 0).
 ULINE AT /C_POS_INICI(C_ANCHO_TOTAL).
ENDIF.
* Inicializamos el valor de la cabecera de la tabla
CLEAR I_ZCLIENXX.

ENDFORM. " MOSTRAR_CLIENTES
&-----
*& Form MOSTRAR_CABECERA_CLIENTES
&-----
* Muestra la cabecera del listado de los clientes existentes *

FORM MOSTRAR_CABECERA_CLIENTES.

* Proceso *

* Hacemos que la línea permanezca fija en el scroll horizontal
NEW-LINE NO-SCROLLING.

```

```
* Activamos el color de cabecera
FORMAT COLOR COL_HEADING INTENSIFIED OFF.
* Escribimos información de cabecera resaltando los valores
* Título y sociedad
WRITE: AT C_POS_TITUL TEXT-009.
WRITE P_BUKRS INTENSIFIED ON.
* Usuario de creación del informe
WRITE AT C_POS_USUAR TEXT-010.
WRITE SY-UNAME INTENSIFIED ON.
* Fecha de creación
WRITE AT C_POS_FECHA TEXT-011.
WRITE SY-DATUM INTENSIFIED ON USING EDIT MASK '___/___/____'.
* Nº de página.
WRITE AT C_POS_PAGNO TEXT-012.
WRITE SY-PAGNO INTENSIFIED ON.
* Escribimos una línea horizontal
ULINE AT /C_POS_INICI(C_ANCHO_TOTAL).
* Activamos el color de cabeceras intensificado
FORMAT COLOR COL_HEADING INTENSIFIED ON.
* Escribimos un espacio en blanco para el campo de marca
WRITE AT /C_POS_INICI '|'.
* Escribimos la cabecera de las columnas separadas por el carácter '|'.
* Hacemos que hasta la columna de nº.cli. sea fija en el scroll horiz.
WRITE: AT C_POS_NCLIE '|' NO-GAP, TEXT-004.
* Hasta la columna hacemos fijo
SET LEFT SCROLL-BOUNDARY.
* Escribimos el detalle de cabecera de cliente
WRITE: AT C_POS_NOMBR '|' NO-GAP, TEXT-005,
 AT C_POS_APEL1 '|' NO-GAP, TEXT-006,
 AT C_POS_APEL2 '|' NO-GAP, TEXT-007,
 AT C_POS_FNACI '|' NO-GAP, TEXT-008,
 AT C_POS_FINAL '|' NO-GAP.
* Escribimos una línea horizontal
ULINE AT /C_POS_INICI(C_ANCHO_TOTAL).
ENDFORM. " MOSTRAR_CABECERA_CLIENTES
```

## **5. Programación de diálogo.**

### **5.1 Introducción**

Los programas de diálogo están orientados al intercambio dinámico de información entre el usuario y el sistema. Un programa de diálogo necesita necesitan.

### **5.2 Module Pool**

Un Module Pool consta de los siguientes elementos:

- Un programa marco que contiene las definiciones de objetos globales, las subrutinas y los módulos de las pantallas. Estos datos suelen estar definidos en programas del tipo INCLUDE que son incluidos en el programa marco.
- Una o varias pantallas (DYNPROS) que se componen de una definición gráfica de los campos de entrada y salida que van a utilizar, y una lógica de proceso que define los procesos que se van a ejecutar antes de la visualización de la pantalla (PBO) y después de la selección de una de las funciones que muestre la pantalla (PAI).
- Uno o varios STATUS que definen las funciones disponibles de cada pantalla en la barra de menús, la barra de símbolos y la barra de pulsadores del sistema.
- Una o varias transacciones que deberán especificar la pantalla inicial que ejecutan. Al ejecutar una transacción del Module Pool se lanza la pantalla asociada y, tras la ejecución de una función por parte del usuario, se realizan los procesos asociados a dicha función y se lanza una nueva pantalla (que puede ser la misma) hasta que se ejecute una función que finalice la ejecución del Module Pool.

A lo largo del tema se realizará un ejemplo práctico de un Module Pool que permitirá dar altas y visualizar el contenido de la tabla de clientes ZCLIENXX creada en el tema del diccionario de datos. Las referencias a este ejemplo práctico aparecerán en cursiva. La pantalla se podrá ir probando con la opción 'Probar' del Screen Painter como se explicará más adelante. Es importante tener en cuenta que después de realizar cambios en la lógica de proceso de una pantalla o en el código del programa se deberán generar los objetos modificados antes de ejecutar la pantalla, ya que sino no se tendrán en cuenta las modificaciones.

### 5.2.1 Programa marco

El programa marco se crea desde el editor ABAP/4.

Ruta de acceso: (En el menú principal de SAP) Herramientas→ Workbench ABAP→Desarrollo→ Editor ABAP(SE38).

Los programas de tipo Module Pool deben comenzar con el literal **'SAPMZ'**.

*Ejemplo práctico: Se creará el programa SAPMZBXX.*

Al seleccionar la opción 'Crear'  aparecerá la pantalla de atributos del programa.

The screenshot shows the 'ABAP: Propiedades de programa SAPMZBXX modificar' dialog box. The title field contains 'Altas y consultas de clientes'. The master language is set to 'ES' (Español). The creator is 'ACRRICOTE' and the creation date is '11.12.2001'. The status is 'Nuevo(revisado)'. In the 'Atributos' section, the type is 'M Modulpool', the status is 'T Programa de test', and the application is '\* Multiplicaciones'. There are checkboxes for 'Bloqueo de editor' (unchecked) and 'Cálculo de coma fija' (checked). The bottom toolbar includes icons for 'Grabar', 'Cancelar', 'Ayuda', 'F5', and 'Salir'.

**Título:** Descripción de la funcionalidad del Module Pool.

*Ej. Altas y consultas de clientes.*

**Tipo:** Determina el tipo de programa .

*Ej. 'M' (Modulepool).*

**Aplicación:** Módulo al que pertenece el programa (FI , HHRR ...).

*Ej. '\*' (Para todas las aplicaciones).*

Grabamos el programa como objeto local. Se creará la cabecera de programa , salimos del editor.

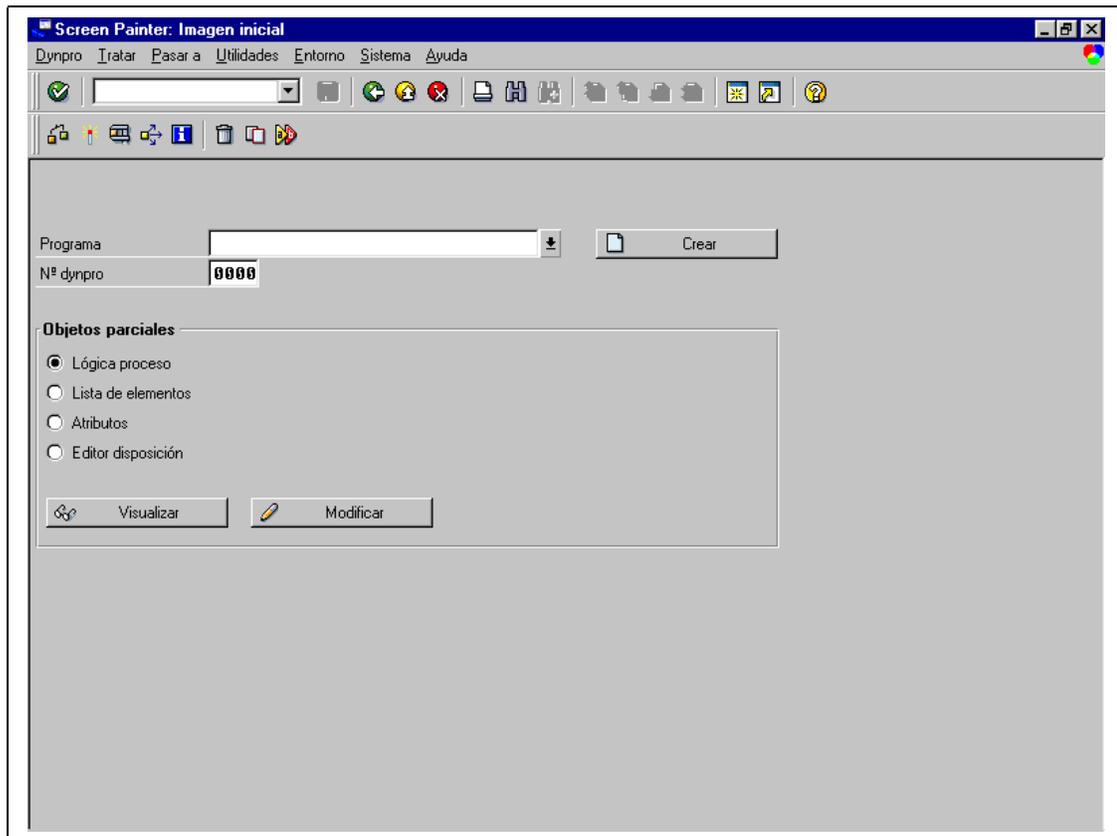
El programa se ha creado sin ninguna sentencia, pero al crear las pantallas se irán creando automáticamente una serie de programas de tipo INCLUDE (que comenzarán con los cinco últimos caracteres del programa marco) que se añadirán automáticamente a nuestro programa a través de las sentencias INCLUDE:

- **MZBXXTOP:** Contendrá las definiciones globales del programa.
- **MZBXXO01:** Contendrá los módulos PBO de las pantallas.
- **MZBXXI01:** Contendrá los módulos PAI de las pantallas.
- **MZBXXF01:** Contendrá las subrutinas del programa. Si nuestro programa no contiene subrutinas, este INCLUDE no estará presente.

### 5.2.2 Atributos de una pantalla

Para crear las pantallas del Module Pool se utiliza el SCREEN PAINTER.

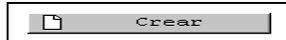
Ruta de acceso: (En el menú principal de SAP) Herramientas → Workbench ABAP → Desarrollo → Interfase usuario → Screen painter (SE51).



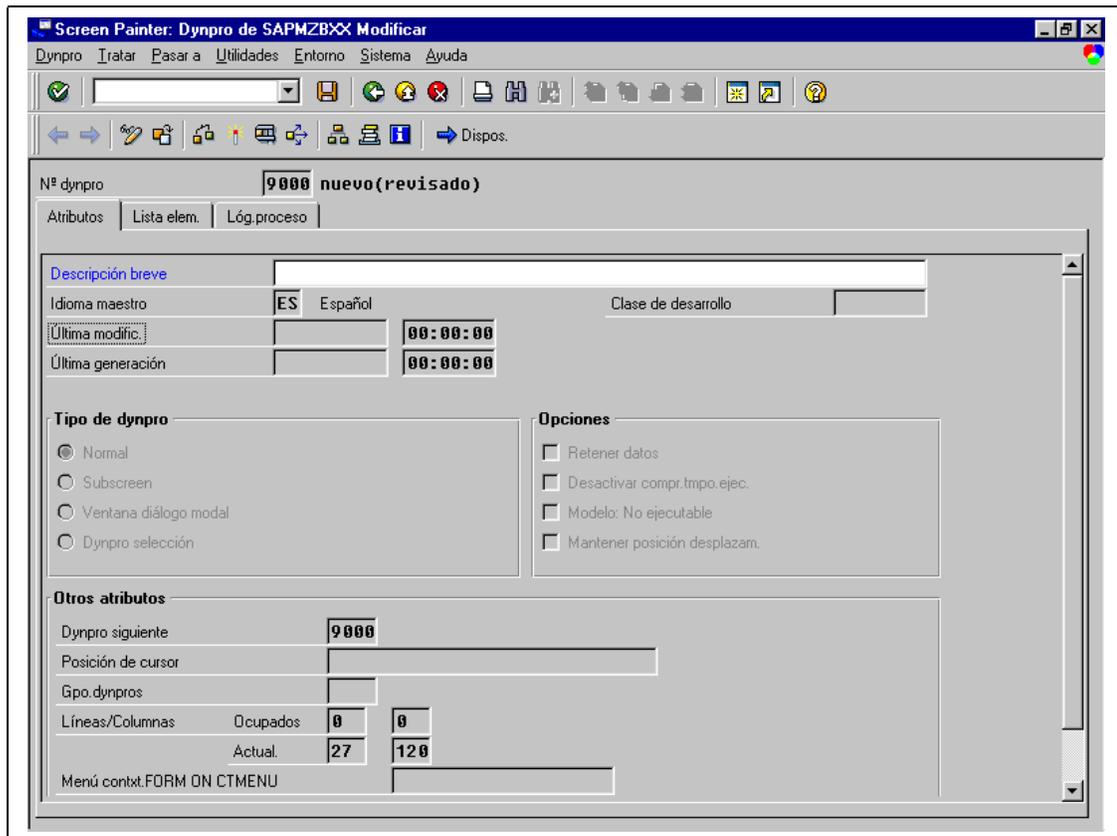
En la pantalla inicial se deberá especificar el nombre del programa marco y el número de la pantalla. Desde esta pantalla se pueden crear, modificar o visualizar todas las partes de una pantalla marcando las distintas opciones de objetos parciales.

*Ej. Se creará la pantalla '9000' en el programa SAPMZBXX marcando la opción. Esta pantalla permitirá dar altas en la tabla ZCLIENXX.*

Al marcar la casilla 'Atributos dynpro' y seleccionar la opción 'Crear'



aparecerá la pantalla para establecer los atributos de la pantalla.



**Descripción breve:** Descripción de la funcionalidad de la pantalla.

Ej. 'Altas de clientes'.

**Tipo dynpro:** Determina el tipo de la pantalla.

Existen los siguientes tipos:

- Normal: Parametrización por defecto.
- Subscreen (Dynpro de include): Este atributo identifica a una pantalla que forma parte de otra pantalla.
- Ventana de diálogo modal: Este tipo de pantallas no tienen barra de menús ni la posibilidad de entradas en el campo de comandos.
- Dynpro selección (vent. Diálogo modal): Esta opción es de utilización interna de SAP para las pantallas de selección generadas automáticamente.

Ej. 'Normal'.

**Otros atributos :**

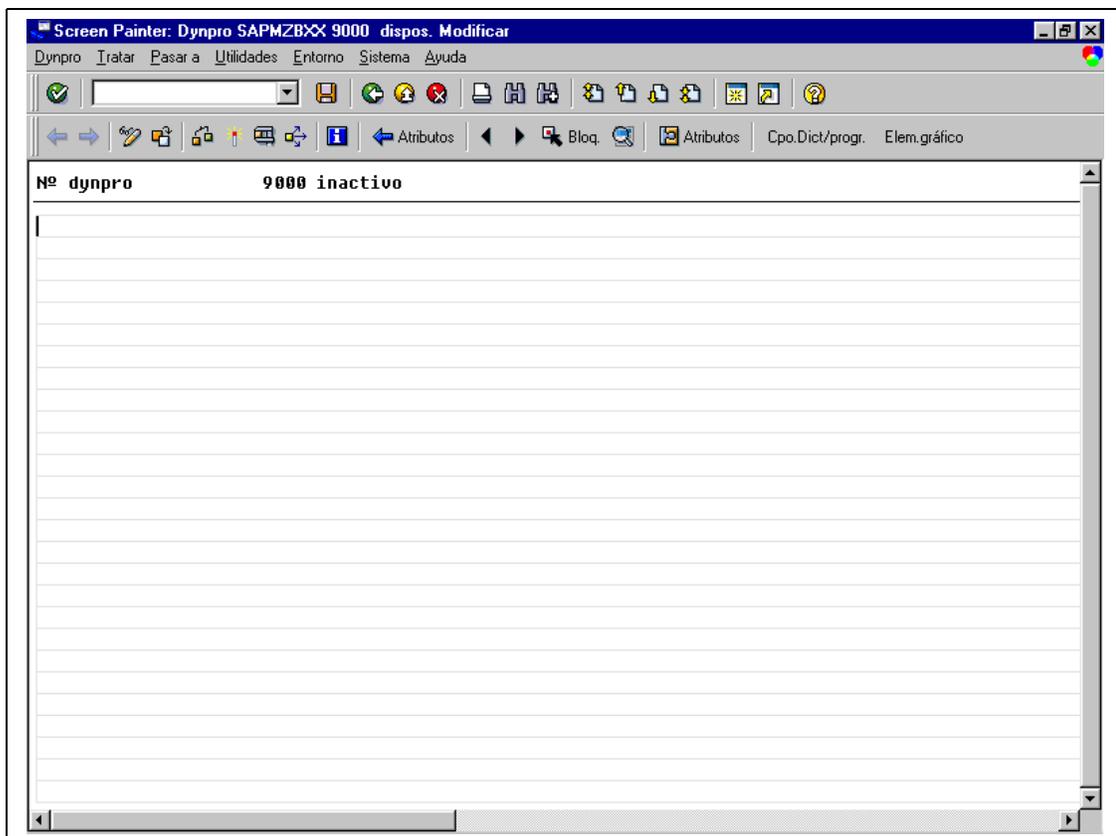
- Dynpro siguiente: Indica el número de la pantalla que se ejecutará automáticamente al finalizar la ejecución de la actual si no se controla específicamente desde la lógica de proceso, presentando por defecto el número de la pantalla actual. El valor '0' finalizará la transacción.

- Posición cursor: En este atributo se puede especificar el nombre del campo en el que aparecerá el cursor al ejecutar la pantalla. Si está en blanco aparecerá en el primero.
- Gpo dynpros: Este atributo permite asignar la pantalla a un grupo de pantallas para poder realizar modificaciones uniformes sobre ellas.
- Lineas/Columnas: Indica las líneas y columnas ocupadas y el número de líneas y columnas de la pantalla.

Una vez finalizada la introducción de atributos se grabarán con el botón  (F11).

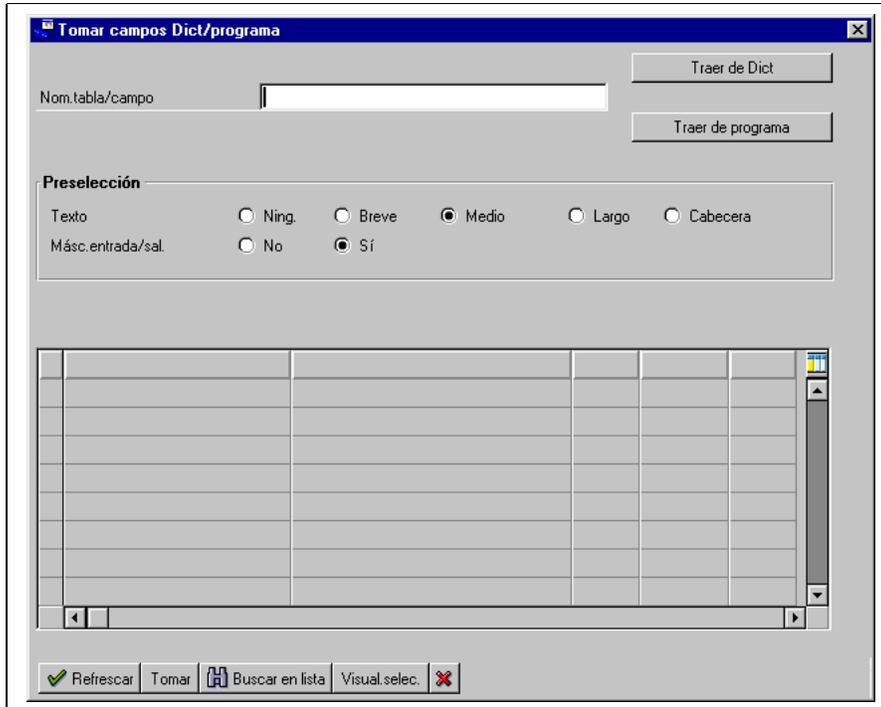
### 5.2.3 Diseño gráfico

Desde la definición de atributos del dynpro seleccionaremos el botón  desde la pantalla de atributos de pantalla aparecerá la pantalla del editor donde se definirá el aspecto de la pantalla.



La opción de menú Pasar a→Campos Dict/programa o botón  permite insertar en la pantalla campos referenciados al diccionario de datos o a variables definidas en el programa.

Ej.: Se crearán los campos donde se podrán introducir los valores que se darán de alta en la tabla de clientes.



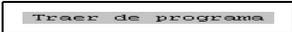
**Nom. Tabla / campo:** En este campo se debe informar el nombre de la variable o del campo de tabla que vamos a incluir en la pantalla. Si se especifica el nombre de una tabla aparecerán todos los campos de la misma pudiendo seleccionarlos posteriormente.

Ej.: 'ZCLIENXX'.

**Preselección:** Permite seleccionar el texto explicativo que acompañará al campo del diccionario y especificar si se creará como campo de entrada/salida.

Ej.: Palabra clave 'Mediano' y Másc. Ed. 'Sí'.

Si se va a insertar una variable definida en el programa se pulsará el botón

 y si es un campo de una tabla del diccionario se pulsará el botón .

Ej.: 'Traer de Dict.'

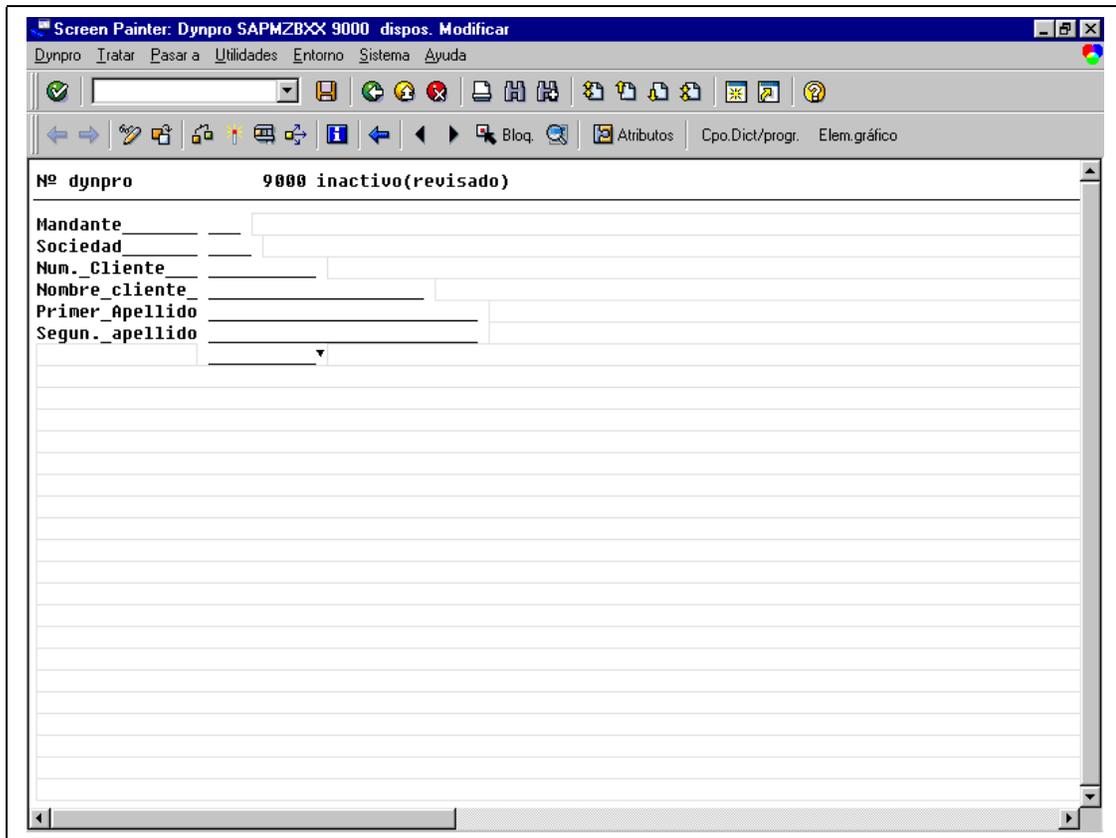
Posteriormente se seleccionará la casilla que aparece a la izquierda del campo que se desea insertar y se pulsa el botón .

Ej.: Se seleccionarán todos.

Una vez recuperado el campo falta posicionarlo en pantalla haciendo doble click en la posición en la que debe aparecer. (En algunos casos aparece siempre pegado al margen izquierdo debido a las opciones de usuario que se tengan definidas, pudiendo moverse posteriormente como se explicará más adelante).

Si se ha insertado un campo del diccionario con palabra clave, se crearán dos campos en la pantalla, uno de texto con la palabra clave, y otro de entrada y salida para visualizar o modificar los datos del campo.

*Ej.: La pantalla quedará de la siguiente forma:*

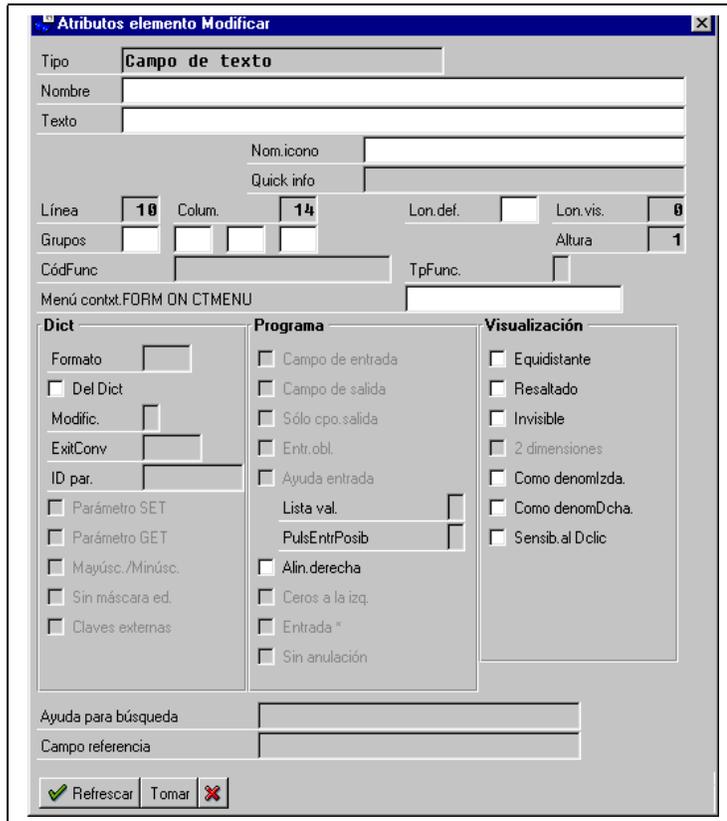


Para mover o borrar los elementos de la pantalla se deberán seleccionar posicionando el cursor en uno de ellos y pulsando el botón  (o haciendo doble click sobre el mismo). Aparecerá seleccionado el campo y se podrá seleccionar un bloque posicionando el cursor en el último campo que se quiera seleccionar y pulsando el botón  (o haciendo doble click sobre el mismo). Una vez que estén seleccionados los campos se podrán borrar pulsando el botón de borrar bloque , o mover posicionando el cursor en la posición deseada y pulsando el botón de desplazar .

La opción de menú Tratar→Crear elemento→... nos permite añadir a la pantalla los elementos que necesitemos en la posición del cursor. Los elementos posibles son los siguientes:

- Palabra clave/Texto: Crea literales informativos en la pantalla. . También se pueden crear escribiendo el texto directamente en la pantalla separando las palabras con el carácter “\_” y pulsando Enter.
- Esquema/entrada: Crea campos de entrada/salida.
- Casilla selección: Crea un campo de entrada de longitud 1 que se puede activar o desactivar en la pantalla. Se podrá comprobar en el programa si la casilla ha sido seleccionada chequeando si la variable asociada contiene el valor ‘X’ (seleccionada) o está vacía (no seleccionada).
- Botón selección: Crea un campo con las mismas características que la casilla de selección con la particularidad de que al agrupar varios botones de selección, solo se puede seleccionar uno de ellos en la pantalla. Para agrupar botones de selección se deberá posicionar el cursor encima del primero y pulsar el botón , después se posicionará el cursor en el último y se pulsará el botón  regresando al editor.
- Pulsador: Crea un botón en la pantalla.
- Marco: Crea un recuadro en la pantalla para agrupar campos visualmente. Para delimitar el ámbito del marco se deberá posicionar el cursor en la posición final del marco y pulsar el botón .
- Table control: Crea una tabla que permite tratar valores almacenados en una tabla.
- Subscreen: Crea una subpantalla.

Al crear los elementos se presentará la pantalla de atributos del elemento.



Dependiendo del tipo de elemento habra atributos que se encuentren desactivados. El significado de los principales atributos es el siguiente:

- Nom.campo: Nombre del campo/variable asociado al elemento de pantalla. Es obligatorio en todos los elementos salvo en campos de texto y en los marcos.
- Txt.campo: Texto de salida asociado al campo. En las casillas y botones de selección se activa este atributo la segunda vez que aparece la pantalla de atributos.
- Icon: Permite asignar un icono al campo informando su nombre. Para seleccionar uno de los disponibles se puede utilizar el matchcode asociado al atributo .
- LongDef.: Longitud del elemento de pantalla.
- Campo entrada: Al activar este atributo la pantalla permitirá introducir datos en el campo.
- Campo salidas: Al estar marcado este atributo se visualizará el contenido del campo en la pantalla.
- Solo salida: Al activar este atributo, el campo de pantalla no admitirá entradas y se visualizará su contenido como un literal.
- Entrada oblig.: Al activar este atributo el campo se definirá de entrada obligatoria, mostrándose un mensaje de error si no se le asigna ningún valor.
- Resaltado: Al activar este atributo el contenido del campo se visualizará resaltado en la pantalla.

- Invisible: Al activar este atributo el los valores introducidos o visualizados en el campo no serán visibles.
- Ayuda de búsqueda: Se puede asociar un objeto Matchcode al campo informando su nombre en este atributo.
- Campo referencia: Este atributo solamente afecta a los campos de tipo CURR (importe de moneda) o QUAN (cantidad). Se especificará la moneda o unidad en la que se formaterán los valores almacenados en el campo.

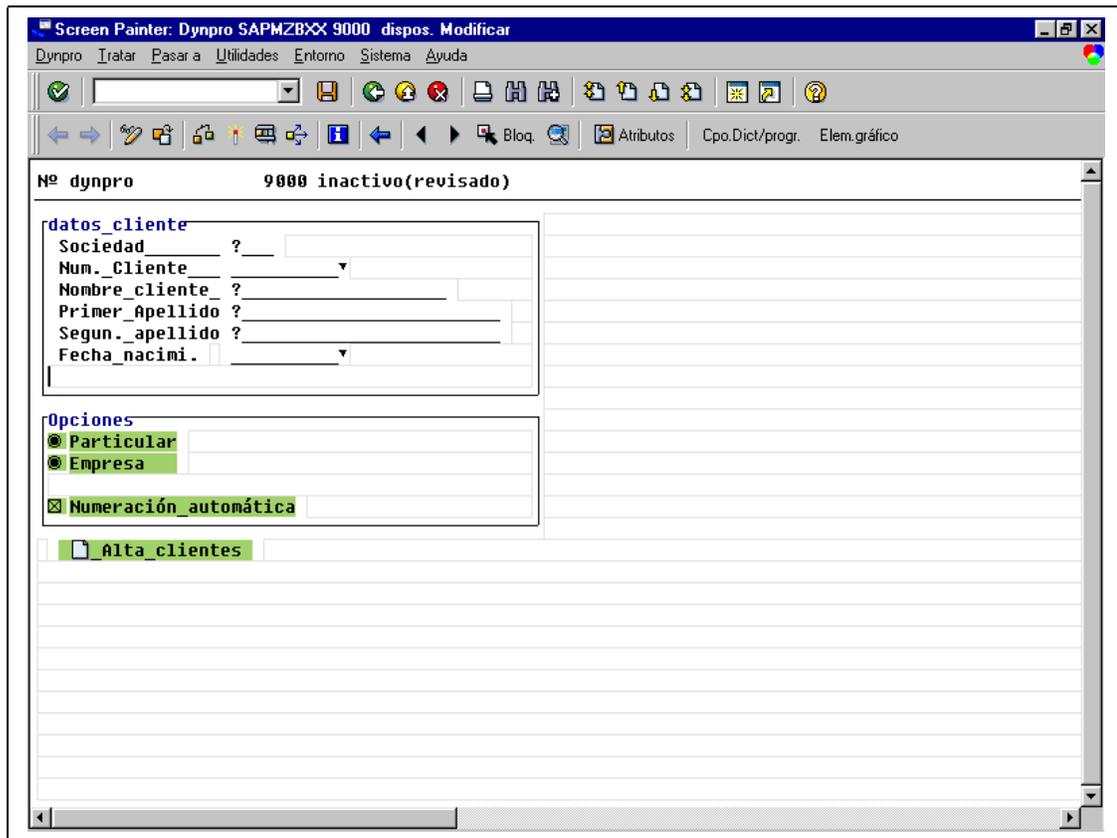
Una vez especificados los atributos del elemento se finaliza la pantalla pulsando el botón .

Ej.: Marcar el atributo 'Entrada oblig' en los campos 'ZCLIENXX-BUKRS', 'ZCLIENXX-NOMBR', 'ZCLIENXX-APEL1', 'ZCLIENXX-APEL2' para establecerlos de entrada obligatoria posicionando el cursor sobre ellos y pulsando el botón



*Crear los siguientes elementos en la pantalla:*

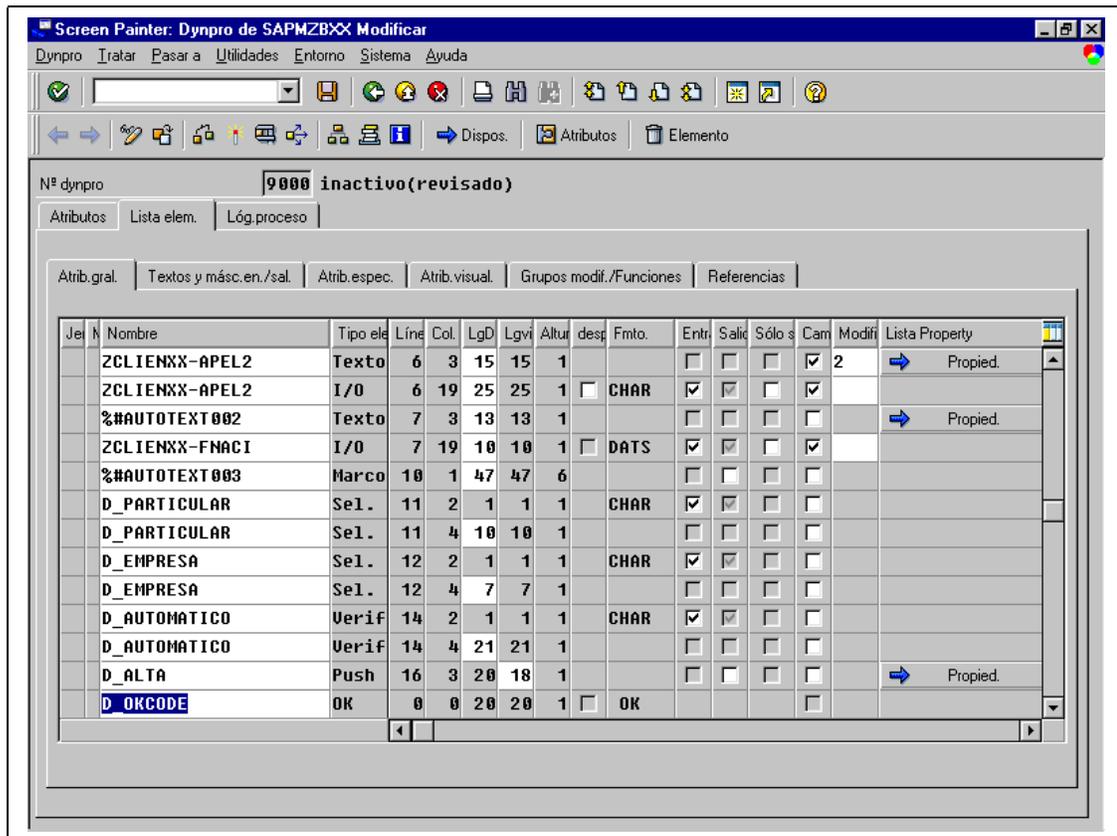
- *Un campo de texto con el texto de campo 'Fec. Nacimiento' posicionada delante del campo ZCLIENXX-NACI como texto explicativo del campo.*
- *Un marco con el texto de campo 'Cliente' que rodee a todos los campos de pantalla para agrupar los campos de entrada que se van a dar de alta.*
- *Un botón de selección con el nombre de campo 'D\_PARTICULAR' y texto de campo 'Particular' que indicará que el cliente es un particular si se selecciona.*
- *Un botón de selección con el nombre de campo 'D\_EMPRESA' y texto de campo 'Empresa' que indicará que el cliente es una empresa si se selecciona.*
- *Se crea un grupo gráfico con los dos botones de selección como se ha descrito anteriormente en la creación de botones de selección, pudiendo elegir una de las dos opciones anteriores según sea el tipo de cliente que se va a dar de alta.*
- *Una casilla de selección con el nombre de campo 'D\_AUTOMATICO' y texto de campo 'Numeración automática' que indicará que el número de cliente se debe informar automáticamente si se selecciona.*
- *Un marco con el texto de campo 'Opciones' que rodee a los nuevos campos de pantalla creados para agrupar las opciones de entrada.*
- *Un pulsador con el nombre de campo 'D\_ALTA', texto de campo 'Alta de cliente', Icono 'ICON\_CREATE' y CódFunc 'ALTA' que provocará el alta de los datos introducidos al ser pulsado.*



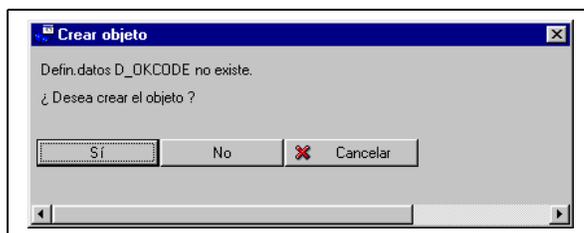
Pulsando el botón  aparecerá una pantalla con todos los campos definidos en la pantalla. **Estos campos deben estar declarados en el INCLUDE MZXXXTOP del programa marco (excepto los literales y los marcos).**

Al final de la lista de campos aparece un campo vacío que se utiliza para asignarle una variable en la que se almacenará el código de función seleccionado por el usuario en la pantalla. Esta variable debe tener la misma estructura que la variable del sistema SY-UCOMM, que también almacena dicho código de función automáticamente.

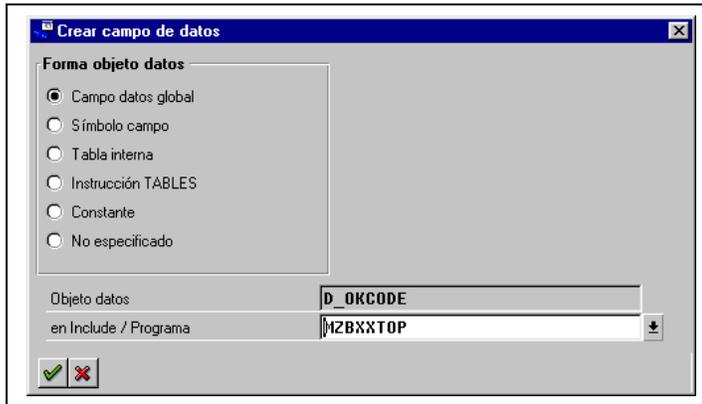
*Ej.: Se asignará la variable D\_OKCODE al campo de pantalla que recibe el código de función seleccionado por el usuario.*



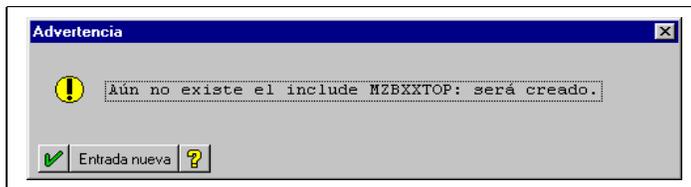
Definiremos la variable automáticamente pulsando doble click sobre ella (sobre el literal D\_OKCODE). El sistema detectará que el módulo aún no ha sido creado y nos propondrá su creación.



Al aceptar la creación de la variable aparecerá una pantalla en la que marcaremos la opción 'campo de datos global' (indicando que se trata de una variable global) e informaremos 'MZBXXTOP' en el campo donde se indica el Include/Programa donde se va a definir la variable.

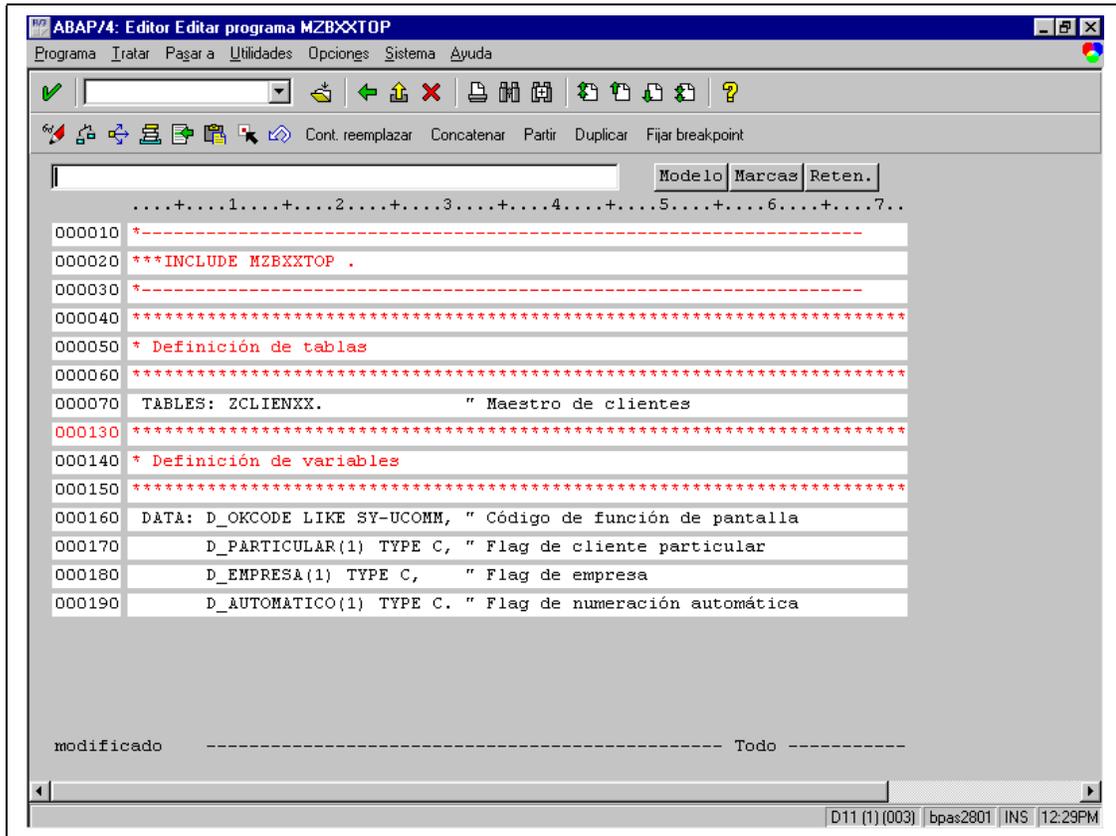


Al pulsar el botón de aceptar, el sistema detecta que el INCLUDE especificado no existe y nos propone su creación (creando la sentencia INCLUDE correspondiente en el programa marco).



Después de pulsar el botón de aceptar asignaremos la estructura del campo SY-UCOMM a la variable creada y documentaremos su funcionalidad.

Se definirán también las variables asociadas a la casilla de selección y los botones de selección como tipo CHAR de longitud 1 y la tabla ZCLIENXX.

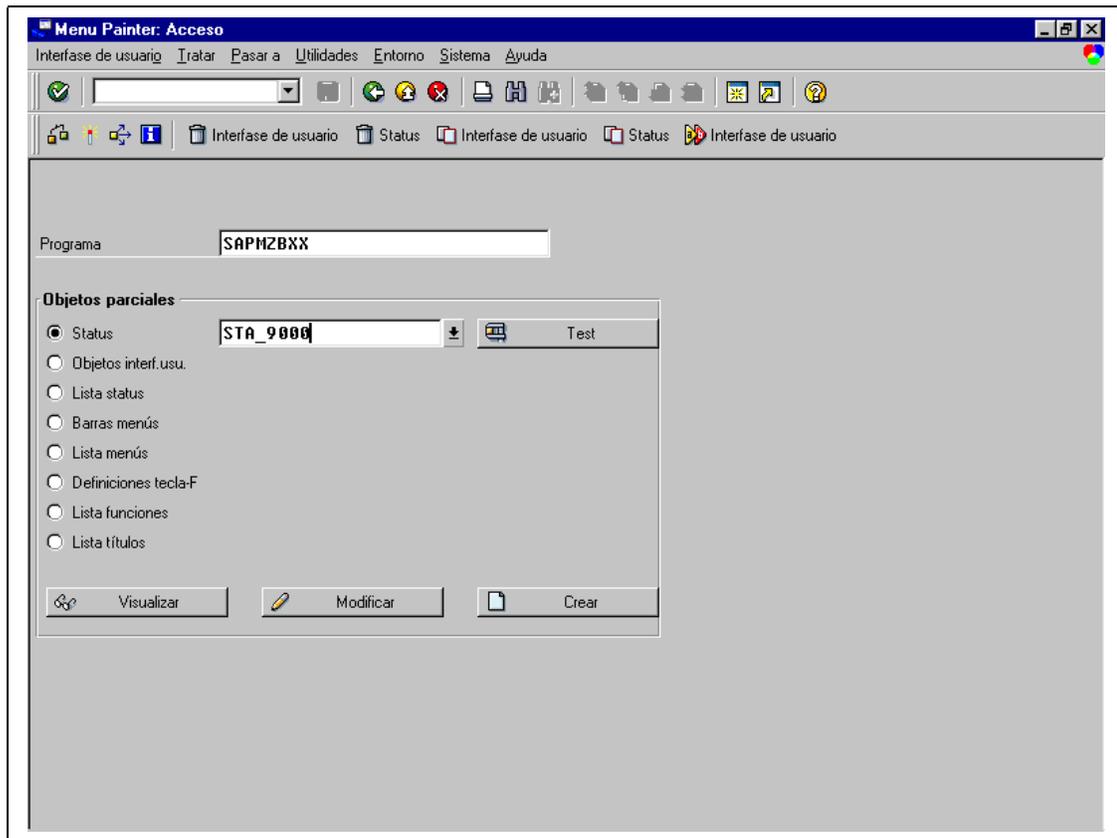


Se finalizará grabando el código  y regresando a la lista de campos .

### 5.2.4 Status de pantalla

Después de realizar el diseño gráfico de una pantalla se debe crear un STATUS que definirá las funciones que tendrá disponibles la pantalla en las barras de menús, de herramientas del sistema y de pulsadores.

Para crear los Status de pantalla del Module Pool se utiliza el MENU PAINTER.  
 Ruta de acceso: (En el menú principal de SAP) Herramientas → Workbench ABAP → Desarrollo → Interfase de usuario → Menu painter (SE41).



En la pantalla inicial se deberá especificar el nombre del programa marco y el del Status que se va a crear. Desde esta pantalla se pueden crear, modificar o visualizar todas las partes de un Status marcando las distintas opciones de objetos parciales. Ej. Se creará el Status 'STA\_9000' en el programa SAPMZBXX marcando la opción 'Status' y pulsando el botón de crear . Este Status definirá las opciones que tendrá disponible la pantalla '9000' de altas de clientes.

Al crear un status aparece una pantalla para definir sus atributos.



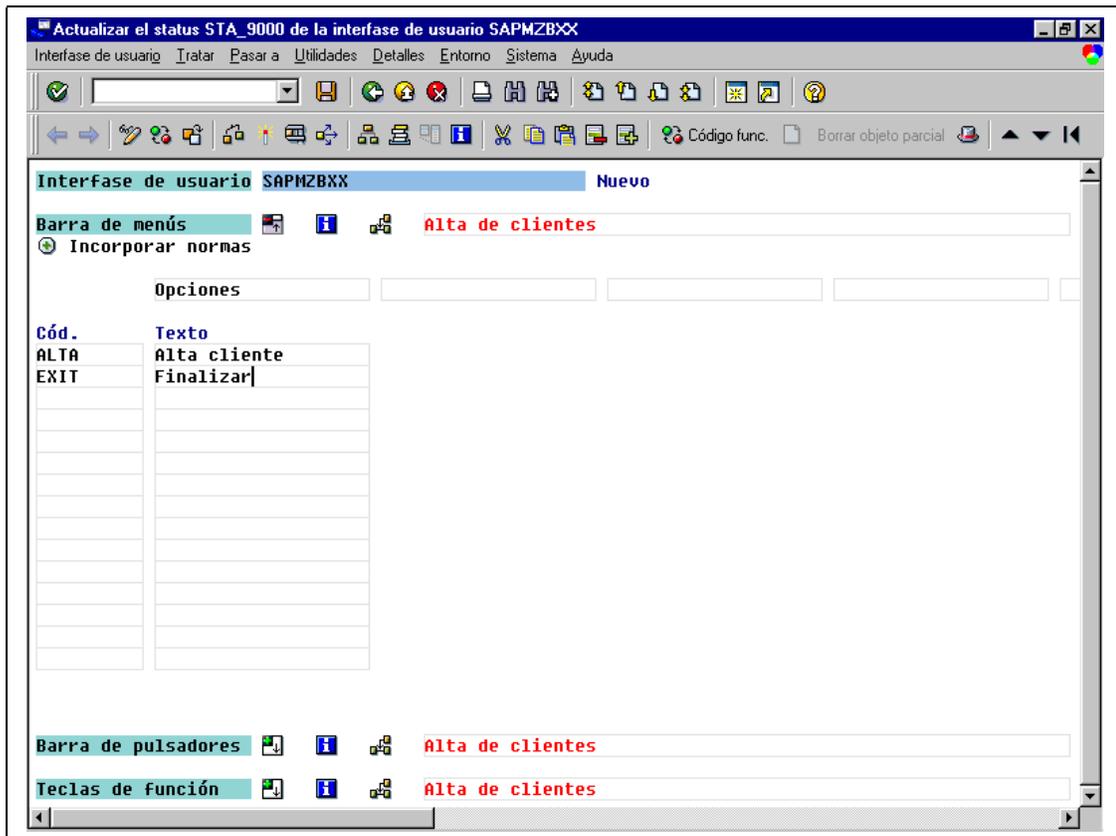
Pulsando el botón de aceptar aparecerá la pantalla de actualización de Status donde definiremos el menú, los símbolos y los pulsadores de la pantalla.

- **Barra de menús.**

Para definir el menú informaremos la descripción del mismo en el campo 'Barra menús'. (Ej.: 'Menú de altas de clientes'), y el texto de las opciones de menú en los campos que aparecen vacíos debajo de la descripción (E.: 'Opciones').

Para informar las opciones de cada una de las opciones del menú , haremos Doble-Click sobre cada una de ellas. Una vez desplegado se crearán las funciones que contendrá la opción de menú informando el código de función asociado (primer campo) y la descripción que aparecerá (segundo campo).

*Ej.: Crearemos la función 'ALTA'-'Alta de cliente' y 'EXIT'-'Finalizar' que realizarán un alta o finalizarán la ejecución de la pantalla respectivamente al ser seleccionadas.*



Se puede crear un submenú informando el campo de descripción y dejando vacío el campo del código de función asociado, desplegándolo posteriormente de la misma forma que las opciones de menú normales.

Para borrar una función de un menú se debe borrar el código de función asociado y la descripción.

- **Barra de teclas de función.**
- Barra de herramientas.  
Sirve para definir las opciones de la barra de herramientas estándar que van a estar disponibles.

En nuestro caso, asociaremos 'BACK', 'EXIT' y 'CANC'.

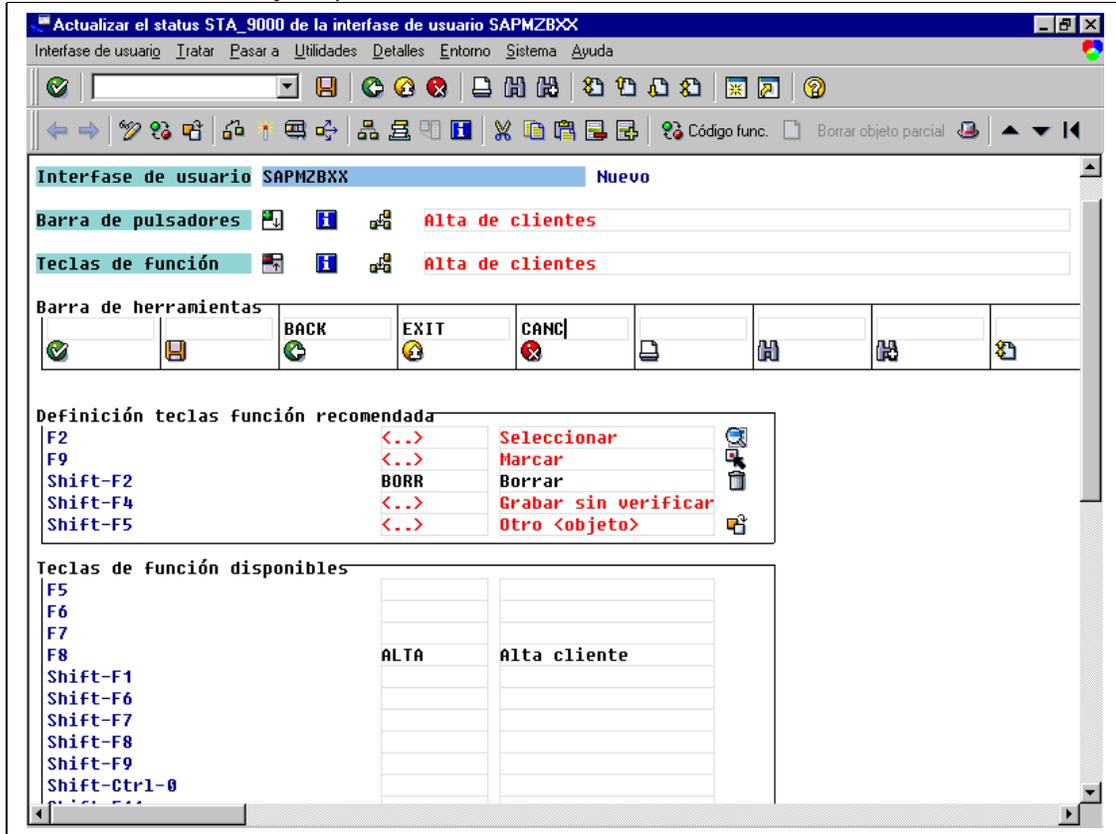
Para crear pulsadores se deberán informar los códigos de función asociados en los campos de cuatro caracteres de la barra de pulsadores.

*Ej.: Crearemos un pulsador para borrar el valor de los campos de pantalla informando el código de función 'BORR' en el primer campo de pulsadores.*

- Teclas de función disponible.

Para asociar una tecla de función a una función del Status se informará el código de función en el campo correspondiente a la tecla de función deseada en 'Definición recomendada teclas func.' o 'Tecla función de libre defin.'.

Ej.: Se asocia el código de función de altas de clientes 'ALTA' a la tecla de función F8 y el código de función de borrado de datos 'BORR' a la tecla de función Shift-F2 con el texto 'Borrar' y se pulsa Enter.

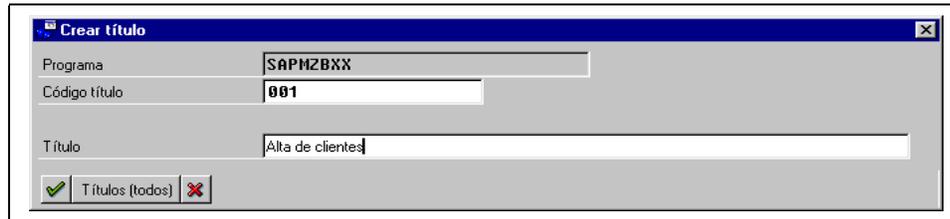


Para eliminar un pulsador se borrará el código de función asociado.

▪ **Barra de pulsadores.**

Definiremos las opciones de la barra de pulsadores que tendrá la pantalla, para ello definiremos , los códigos de función indicado en el apartado de lista de funciones.(En nuestro caso indicaremos u único botón correspondiente a la opción de borrar).

Otro componente necesario para la composición de pantallas es la definición de títulos de pantalla. Para crear / visualizar los títulos en el menú de acceso al MENU PAINTER seleccionamos la opción 'Lista de títulos' y seleccionamos la opción de crear , nos aparecerá la siguiente pantalla donde informar el nombre y descripción del título.



Ej.: Crearemos el título 001 'Altas de clientes'.

Nos aseguramos de activar tanto el estatus definido como el título. (  ).

### 5.2.5 Lógica de proceso

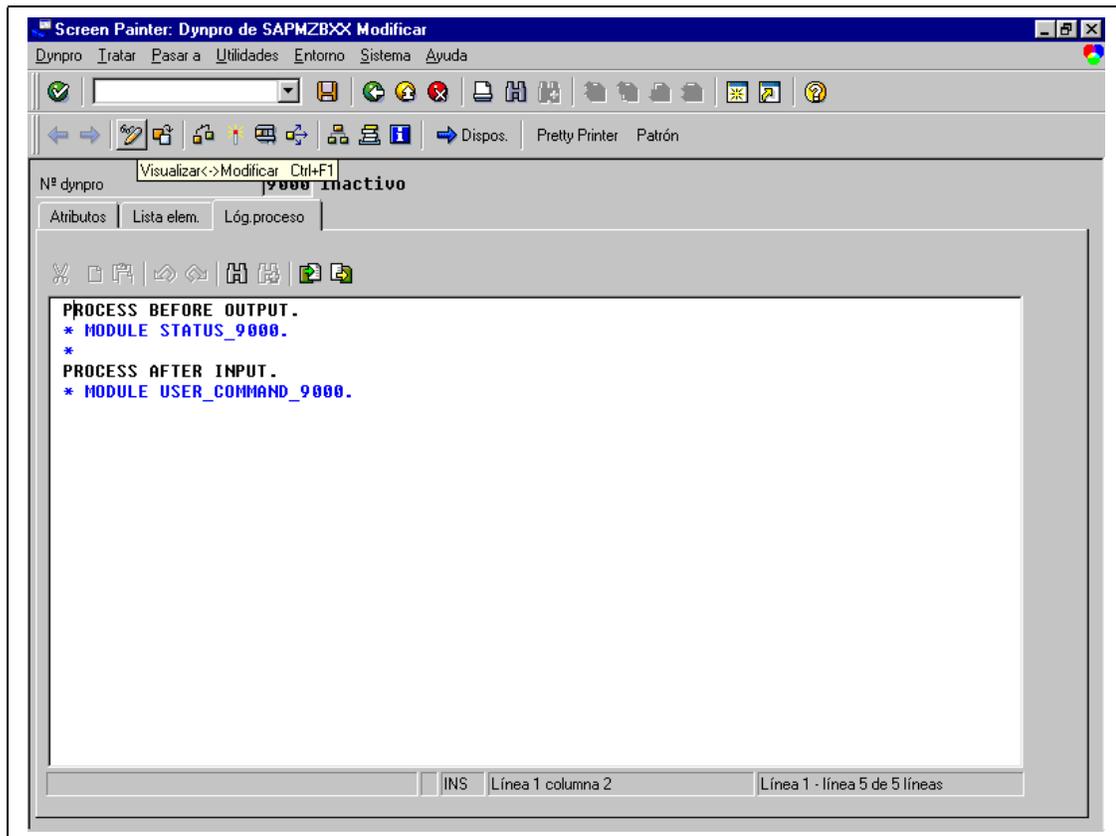
Las instrucciones ABAP/4 que se ejecutarán antes de visualizar una pantalla o cuando el usuario selecciona una función de la pantalla se definen en la lógica de proceso.

La programación de la lógica de proceso se estructura en cuatro eventos de pantalla:

- **PROCESS BEFORE OUTPUT:** Sentencias que se procesan antes de la visualización de la pantalla.
- **PROCESS AFTER INPUT:** Sentencias que se ejecutan después de que el usuario haya seleccionado una función de la pantalla.
- **PROCESS ON HELP-REQUEST:** Sentencias que se ejecutan cuando el usuario presiona la tecla de función F1 (Ayuda) en un campo de la pantalla.
- **PROCESS ON VALUE-REQUEST:** Sentencias que se ejecutan cuando el usuario presiona la tecla de función F4 (Entradas posibles) en un campo de la pantalla.

Para crear/modificar la lógica de proceso de una pantalla se seleccionará la opción 'Lógica proceso' y se pulsará el botón de modificar  en la pantalla inicial del Screen Painter.

Por defecto se crean los eventos PROCESS BEFORE OUTPUT (PBO) y PROCESS AFTER INPUT (PAI).



Las pantallas deben tener asignado un título y un Status que definen el título, los menús, la barra de herramientas estándar y pulsadores que tendrán activos. Para realizar la asignación se utilizarán las siguientes sentencias dentro de un módulo en el evento PBO, ya que no se pueden utilizar directamente en la lógica de proceso de la pantalla:

- SET PF-STATUS <nombre>: Asigna un Status a la pantalla pudiendo excluir funciones con la cláusula EXCLUDING.
- SET TITLEBAR <número>: Asigna un título a la pantalla.

Para provocar el final de la ejecución de un Module Pool se utiliza la sentencia LEAVE PROGRAM.

Las sentencias que se pueden utilizar directamente en la lógica de proceso de una pantalla son las siguientes:

- *Sentencia MODULE.*  
Las sentencias de la lógica de proceso se suelen estructurarse en módulos delimitados por las sentencias MODULE y ENDMODULE. Estos módulos se almacenan en programas INCLUDE creados con el editor ABAP/4 y en ellos se pueden utilizar todas las sentencias ABAP/4.

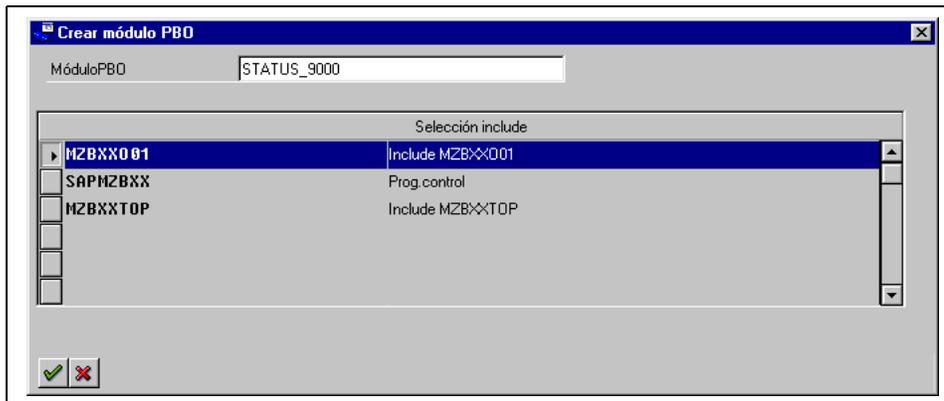
Para ejecutar los módulos desde la lógica de proceso del editor Screen Painter se utiliza la sentencia MODULE <nombre>.

Para crear un módulo desde la lógica de proceso se crea la sentencia de llamada y se realiza un Doble-click sobre el nombre del módulo para que el sistema lo cree automáticamente en el programa INCLUDE correspondiente, que también será creado si es necesario.

Ej.: Crearemos el módulo 'STATUS\_9000' (que aparece por defecto en el evento PBO de la lógica de proceso) para asignar un Status y un título a la pantalla. Se borra el asterisco que aparece al inicio de la línea, se graba la lógica de proceso pulsando el icono de grabar  y se hace doble click sobre el literal 'STATUS\_9000'.

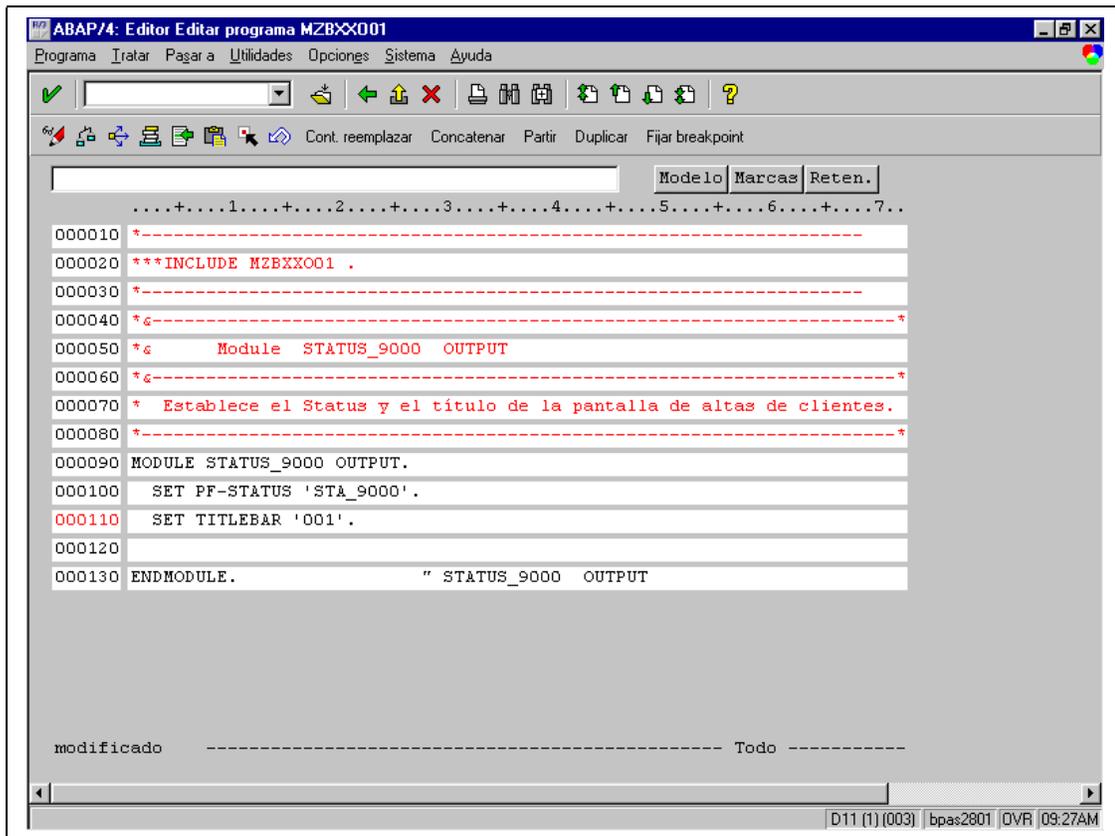
*El sistema detectará que el módulo aún no ha sido creado y nos propondrá su creación.*

*Al aceptar la creación del módulo, el sistema nos propondrá la creación de un INCLUDE nuevo en el que se almacenarán todos los módulos PBO que se creen en el module pool.*



*Al aceptar la creación del include, el sistema creará automáticamente la sentencia INCLUDE <nombre> en el programa marco.*

*Después de pulsar el botón de aceptar documentaremos la funcionalidad del módulo en el editor ABAP/4 y eliminamos el asterisco de las dos sentencias que aparecen por defecto asignando el Status 'STA\_9000' y el título '001' (creados anteriormente en el Menu Painter).*



Al finalizar se grabará el código, se regresará a la lógica de proceso de la pantalla y se creará un nuevo modulo PBO 'INICIALIZAR\_9000' (creando la llamada con la sentencia MODULE después de la llamada al módulo 'STATUS\_9000' y haciendo Doble- Click sobre el nombre), que inicializará el campo de sociedad con el valor por defecto '0001'.

```

&-----
*& Module INICIALIZAR_9000 OUTPUT
&-----
* Inicializa los campos de la pantalla de altas de clientes.

MODULE INICIALIZAR_9000 OUTPUT.

* Se establece el valor por defecto de la sociedad si no está informada
 IF ZCLIENXX-BUKRS IS INITIAL.
 ZCLIENXX-BUKRS = C_BUKRS.
 ENDIF.

ENDMODULE. " INICIALIZAR_9000 OUTPUT

```

La constante C\_BUKRS se deberá definir en el INCLUDE MZBXXTOP con la estructura del campo ZCLIENXX-BUKRS y el valor '0001' de la siguiente forma:

```

* Definición de constantes

CONSTANTS: C_BUKRS LIKE ZCLIENXX-BUKRS " Sociedad por defecto
 VALUE '0001'.
```

Se creará un nuevo modulo PAI 'USER\_COMMAND\_9000' que aparece por defecto en el evento PAI (quitándole el asterisco, haciendo doble click sobre el nombre y creando el nuevo INCLUDE 'MZBXXI01' para los módulos PAI), que ejecutará las sentencias asociadas a la función de pantalla seleccionada por el usuario.

```
&-----
*& Module USER_COMMAND_9000 INPUT
&-----
* Ejecuta las sentencias asociadas al código de función devuelto por
* la pantalla de altas de clientes.

```

```
MODULE USER_COMMAND_9000 INPUT.
```

```
CASE D_OKCODE.
 WHEN 'ALTA'. " Alta de cliente
 PERFORM ALTA_CLIENTE.
 WHEN 'BORR'. " Borrado de datos de cliente
 PERFORM BORRAR_DATOS.
ENDCASE.
```

```
* Se inicializa el valor de retorno de la pantalla
CLEAR D_OKCODE.
```

```
ENDMODULE. " USER_COMMAND_9000 INPUT
```

Se creara la subrutina 'ALTA\_CLIENTE' (haciendo doble click sobre el nombre y creando el nuevo INCLUDE 'MZBXXF01' para las subrutinas), que creará una entrada en la tabla de clientes con los datos introducidos en los campos de pantalla.

```
&-----
*& Form ALTA_CLIENTE
&-----
* Añade una entrada a la tabla de clientes ZCLIENXX con los datos
* introducidos por pantalla.

```

```
FORM ALTA_CLIENTE.
```

```
* Se comprueba que el numero de cliente se informa correctamente
PERFORM NUMERO_CLIENTE.
```

```
* Se añade el registro de cabecera, ya informado con los datos del
```

\* nuevo cliente, a la tabla de clientes.

INSERT ZCLIENXX.

\* Se comprueba que la inserción se ha realizado correctamente.

IF SY-SUBRC <> '0'.

\* Error al insertar el registro

MESSAGE E000(38) WITH TEXT-001.

ELSE.

\* El cliente ha sido dado de alta

MESSAGE S000(38) WITH TEXT-002.

ENDIF.

ENDFORM. " ALTA\_CLIENTE

Se creará la subrutina 'NUMERO\_CLIENTE', que comprueba que el número de cliente se informa correctamente y lo rellena con ceros a la izquierda.

```

*& Form NUMERO_CLIENTE
&-----
* Comprueba que el número de cliente se informa correctamente

```

FORM NUMERO\_CLIENTE.

DATA: L\_NCLIE LIKE ZCLIENXX-NCLIE. " Número de cliente.

IF ZCLIENXX-NCLIE IS INITIAL.

\* Numeración automática: Se selecciona el último número de cliente

\* dado de alta

SELECT NCLIE INTO ZCLIENXX-NCLIE

FROM ZCLIENXX

WHERE BUKRS = ZCLIENXX-BUKRS.

ENDSELECT.

IF SY-SUBRC = 0.

\* Se informa el número de cliente con el posterior al seleccionado

ZCLIENXX-NCLIE = ZCLIENXX-NCLIE + 1.

ELSE.

\* No existen clientes dados de alta, se inicializa el número

ZCLIENXX-NCLIE = 1.

ENDIF.

ELSE.

\* Numeración manual: Se comprueba si ya existe un cliente dado de alta

\* con el número introducido

SELECT SINGLE NCLIE INTO L\_NCLIE

FROM ZCLIENXX

WHERE BUKRS = ZCLIENXX-BUKRS

AND NCLIE = ZCLIENXX-NCLIE.

IF SY-SUBRC = 0.

- \* Se inicializa el código de función de la pantalla para que no se vuelva a ejecutar el alta al pulsar ENTER después del error.
- \* CLEAR D\_OKCODE.
- \* Error: El número de cliente ya existe.
- MESSAGE E000(38) WITH TEXT-003.
- ENDIF.

ENDIF.

- \* Se rellena el número de cliente con ceros por la izquierda
- UNPACK ZCLIENXX-NCLIE TO ZCLIENXX-NCLIE.

ENDFORM. " NUMERO\_CLIENTE

Se crearán los elemento de texto TEXT-001 y TEXT-002 en el programa marco con los textos 'Error al insertar el registro' y 'El cliente ha sido dado de alta' respectivamente.

Se creará la subrutina 'BORRAR\_DATOS', que borrará los datos de los campos de pantalla.

```

*& Form BORRAR_DATOS
&-----
* Borra los datos de los campos de pantalla.

```

FORM BORRAR\_DATOS.

- \* Se borran los datos del cliente.
- CLEAR: ZCLIENXX.

- \* Se establecen los valores por defecto de las opciones de cliente
- D\_PARTICULAR = 'X'. " Se marca la opción de particular por defecto
- CLEAR: D\_EMPRESA,
- D\_AUTOMATICO.

ENDFORM. " BORRAR\_DATOS

La sentencia MODULE dispone las siguientes cláusulas, que solo tienen sentido en el evento PAI, que condicionan su ejecución al tipo de función seleccionado en la pantalla:

- AT EXIT-COMMAND: Provoca que solamente se ejecute el módulo cuando se seleccione una función de pantalla del tipo 'E' (Comando Exit). Al seleccionar una función de este tipo, el sistema no ejecutará los módulos que no tengan esta cláusula y tampoco ejecutará chequeos como los que realiza con los campos de

entrada obligatoria, pasando directamente a ejecutar los módulos que tengan la cláusula AT EXIT-COMMAND. Esta cláusula se suele utilizar para los módulos que controlan las funciones de salida de las pantallas, permitiendo finalizar su ejecución sin realizar ninguna verificación sobre los valores que contengan sus campos.

- AT CURSOR-SELECTION: Provoca que solamente se ejecute el módulo cuando se ejecute sobre un campo una función de selección asociada a la tecla de función F2 o se haga doble click sobre él.

*Ej.: Se creará el módulo PAI 'SALIR\_9000' con la cláusula AT EXIT-COMMAND, que finalizará la ejecución del programa cuando se seleccione una función del tipo 'E' (Comando Exit).*

```

&-----
*& Module SALIR_9000 INPUT
&-----
* Ejecuta las sentencias asociadas a los código de función del tipo
* EXIT COMMAND en la pantalla 9000.

MODULE SALIR_9000 INPUT.

 CASE D_OKCODE.
 * Regresar, Finalizar y Cancelar
 WHEN 'BACK' OR 'EXIT' OR 'CANC'.
 LEAVE PROGRAM.
 ENDCASE.

 ENDMODULE. " SALIR_9000 INPUT

```

*Se modifican las funciones 'BACK', 'EXIT' y 'CANC' del Status 'STA\_9000' desde la opción de menú 'Pasar a → Lista funciones' del Menu Painter, estableciéndolas de tipo 'E' (Comando Exit), grabando y activando el Status.*

El envío de un mensaje dentro de un módulo provocará diferentes efectos en el programa dependiendo del tipo de mensaje:

- Mensaje de error (tipo 'E'): El sistema volverá a presentar inmediatamente la pantalla sin procesar el evento PBO y mostrará el mensaje sin permitir realizar entradas en ningún campo. Al seleccionar una función de pantalla se volverá a procesar directamente el módulo que contiene el error a no ser que se seleccione una función del tipo 'E', en cuyo caso se ejecutarán los módulos que tengan la cláusula AT EXIT-COMMAND.
- Mensaje de advertencia (tipo 'W'): El sistema volverá a presentar inmediatamente la pantalla sin procesar el evento PBO y mostrará el mensaje sin permitir realizar entradas en ningún campo. Si se pulsa Enter seguirá con la ejecución del programa desde el punto donde se ha enviado el mensaje, sino se comportará como un mensaje de error y volverá a procesar directamente el módulo que contiene el error.
- Mensaje informativo (tipo 'I'): Se detiene el proceso de la pantalla y se muestra el mensaje en una pantalla de diálogo. Al presionar Enter se continua con el proceso en la sentencia siguiente al envío del mensaje.

- Mensaje de suceso (tipo 'S'): El mensaje se mostrará en la siguiente pantalla que se procese.
- Mensaje de cancelación (tipo 'A'): Se detiene el proceso de la pantalla y se muestra el mensaje en una pantalla de diálogo. Al presionar Enter finaliza la ejecución del programa.
- *Sentencia FIELD.*  
Esta sentencia permite asociar la ejecución de un módulo a un campo de pantalla permitiendo utilizar las siguientes cláusulas (que solamente tienen sentido en el evento PAI) con la sentencia MODULE:
  - ON INPUT: Provoca que solamente se ejecute el módulo si el campo asociado contiene un valor distinto al inicial en función del tipo de dato (por ejemplo espacios en los campos de tipo carácter).
  - ON REQUEST: Provoca que solamente se ejecute el módulo si el campo asociado ha sido modificado.
  - ON -INPUT: Provoca que solamente se ejecute el módulo si el campo asociado contiene el valor '\*' y además tiene activo el atributo de campo '\* Entry'.

Al mostrar un mensaje de error o advertencia dentro de un módulo asociado a un campo de pantalla, el sistema volverá a presentar inmediatamente la pantalla sin procesar el evento PBO y mostrará el mensaje permitiendo eliminar el error modificando el valor del campo sin permitir modificar ningún otro campo.

*Ej.: Se creará el módulo PAI 'CHEQUEAR\_SOCIEDAD' asociado al campo 'ZCLIENXX-BUKRS' al inicio del evento PAI, que chequeará que la sociedad introducida existe en la tabla standard de sociedades 'T001' siempre que se modifique su valor. Se deberá declarar la tabla de sociedades 'T001' en el INCLUDE 'MZBXXTOP'.*

*Lógica de proceso:*

*PROCESS AFTER INPUT.*

*\* Chequeo de la sociedad*

*FIELD ZCLIENXX-BUKRS*

*MODULE CHEQUEAR\_SOCIEDAD ON REQUEST.*

*Include MZBXXI01:*

*\*&-----\**

*\*& Module CHEQUEAR\_SOCIEDAD INPUT*

*\*&-----\**

*\* Chequea que la sociedad introducida existe en la tabla de sociedades.*

*\*-----\**

*MODULE CHEQUEAR\_SOCIEDAD INPUT.*

*\* Se selecciona la sociedad en la tabla de sociedades*

*SELECT SINGLE \* FROM T001*

*WHERE BUKRS = ZCLIENXX-BUKRS.*

*IF SY-SUBRC <> 0.*

```
* Error: La sociedad & no está prevista.
MESSAGE E165(F5) WITH ZCLIENXX-BUKRS.
ENDIF.
```

```
ENDMODULE. " CHEQUEAR_SOCIEDAD INPUT
```

```
Include MZBXXTOP:
```

```

```

```
* Definición de tablas
```

```

```

```
TABLES: ZCLIENXX, " Maestro de clientes
 T001. " Sociedades
```

El transporte de los valores introducidos en los campos de pantalla se actualizan en las variables asociadas del programa al inicio del evento PAI, excepto los campos que tienen asociada una sentencia FIELD. En este caso su contenido no se actualiza con el introducido en la pantalla hasta que se ejecute la sentencia FIELD correspondiente. Por este motivo no se deberá utilizar el valor de un campo que tenga una sentencia FIELD asociada, en sentencias del programa que se ejecuten antes que ésta, ya que se estaría utilizando el valor antiguo del campo.

- *Sentencias CHAIN ... ENDCHAIN.*  
Estas sentencias permiten agrupar varias sentencias FIELD asociando la ejecución de un módulo a varios campos y permitiendo utilizar las siguientes cláusulas con la sentencia MODULE:
  - ON INPUT: Solamente se ejecuta el módulo si todos los campos asociados contienen un valor distinto al inicial en función del tipo de dato
  - ON CHAIN-INPUT: Solamente se ejecuta el módulo si alguno de los campos asociados contiene un valor distinto al inicial en función del tipo de dato.
  - ON REQUEST: Solamente se ejecuta el módulo si todos los campos asociados han sido modificados.
  - ON CHAIN-REQUEST: Solamente se ejecuta el módulo si algún campo asociado ha sido modificado.

*Ej.: Se creará el modulo PAI 'CHEQUEAR\_AUTOMATICO' asociado a los campos 'ZCLIENXX-NCLIE' y 'D\_AUTOMATICO' llamándolo después del módulo 'CHEQUEAR\_SOCIEDAD', que chequeará que no esté informado el numero de cliente si está marcado el flag de numeración automática y que obligará a informarlo en caso contrario.*

*Lógica de proceso:*

```
* Chequeo de numeración automática de clientes
```

```
CHAIN.
```

```
FIELD: ZCLIENXX-NCLIE,
 D_AUTOMATICO MODULE CHEQUEAR_AUTOMATICO.
ENDCHAIN.
```

```
Include MZBXXI01:
```

```

&-----
*& Module CHEQUEAR_AUTOMATICO INPUT
&-----
* Chequea que el numero de cliente y el flag de numeración automática
* no estan informados o vacíos simultáneamente.

MODULE CHEQUEAR_AUTOMATICO INPUT.

 IF D_AUTOMATICO IS INITIAL.
 * Flag de numeración automática desmarcado
 IF ZCLIENXX-NCLIE IS INITIAL.
 * Número de cliente no informado
 * Error: Informe el número de cliente o marque el flag de num. aut.
 MESSAGE E000(38) WITH TEXT-004 TEXT-005.
 ENDIF.
 ELSE.
 * Flag de numeración automática marcado
 IF NOT ZCLIENXX-NCLIE IS INITIAL.
 * Número de cliente informado
 * Error: Borre el número de cliente o desmarque el flag de num. aut.
 MESSAGE E000(38) WITH TEXT-006 TEXT-005.
 ENDIF.
 ENDIF.

ENDMODULE. " CHEQUEAR_AUTOMATICO INPUT

```

Se crearán los elementos de texto TEXT-004, TEXT-005 y TEXT-006 en el programa marco con los textos 'Informe el número de cliente o marque el flag', 'de numeración automática', 'Borre el número de cliente o desmarque el flag' respectivamente.

Ej.: Se creará el modulo PAI 'CHEQUEAR\_TIPO\_CLIENTE' asociado a los campos 'ZCLIENXX-FNACI, 'D\_PARTICULAR' y 'D\_EMPRESA' llamándolo después del módulo 'CHEQUEAR\_AUTOMATICO', que chequeará que la fecha de nacimiento está informada si el cliente es un particular y que no lo está si es una empresa.

Lógica de proceso:

```

* Chequeo de tipo de cliente
CHAIN.
 FIELD: ZCLIENXX-FNACI,
 D_PARTICULAR,
 D_EMPRESA MODULE CHEQUEAR_TIPO_CLIENTE.
ENDCHAIN.

```

Include MZBXXI01:

```

&-----
*& Module CHEQUEAR_TIPO_CLIENTE INPUT
&-----

```

*\* Chequea que la fecha de nacimiento está informada si el cliente es un particular y que no lo está si es una empresa.*

*\*-----\**  
MODULE CHEQUEAR\_TIPO\_CLIENTE INPUT.

IF D\_EMPRESA IS INITIAL.

*\* Cliente tipo particular*

IF ZCLIENXX-FNACI IS INITIAL.

*\* Fecha de nacimiento de cliente no informada*

*\* Error: Los particulares deben tener informada la f. de nacimiento*

MESSAGE E000(38) WITH TEXT-007 TEXT-008.

ENDIF.

ELSEIF NOT ZCLIENXX-FNACI IS INITIAL.

*\* Cliente tipo empresa con la fecha de nacimiento informada*

*\* Error: Las empresas no deben tener informada la f. de nacimiento*

MESSAGE E000(38) WITH TEXT-009 TEXT-008.

ENDIF.

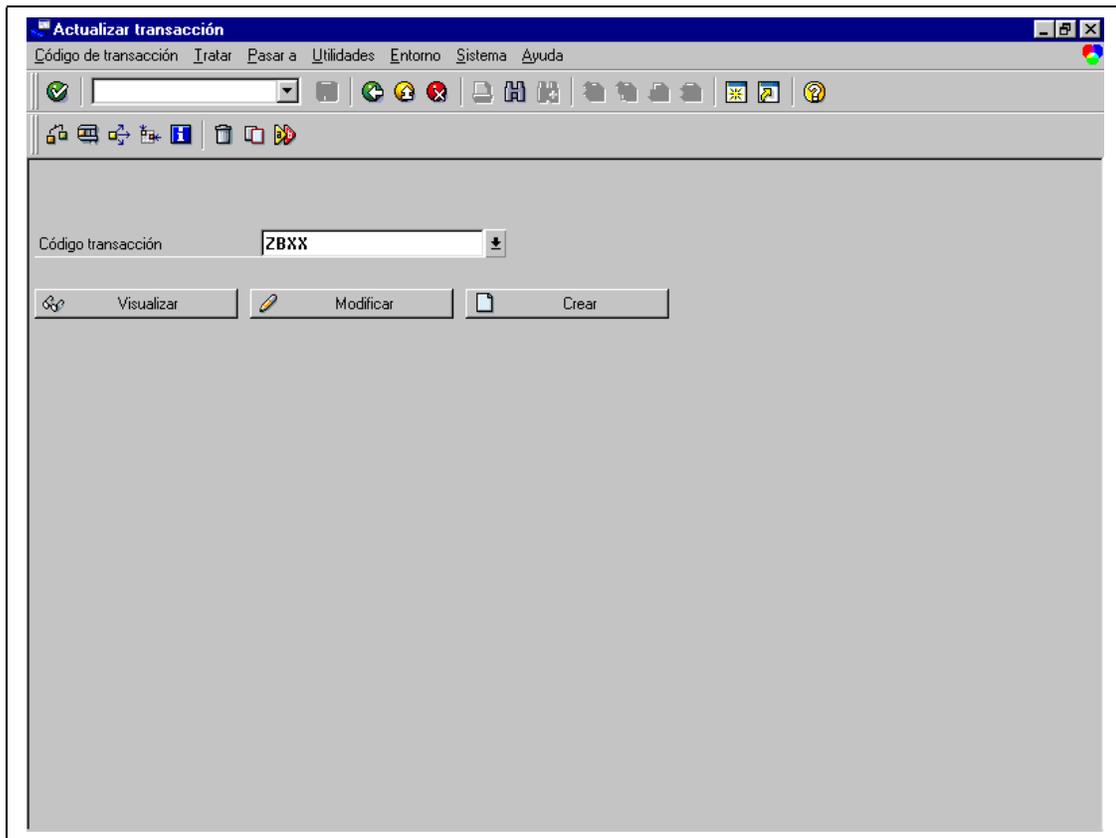
ENDMODULE. " CHEQUEAR\_TIPO\_CLIENTE INPUT

*Se crearán los elementos de texto TEXT-007, TEXT-008 y TEXT-009 en el programa marco con los textos 'Los particulares deben tener informada', 'la fecha de nacimiento', 'Las empresas no deben tener informada' respectivamente.*

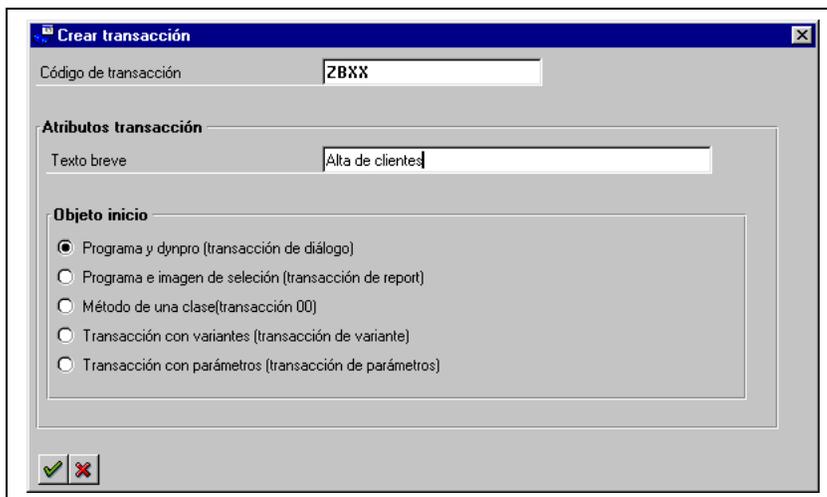
### **5.2.6 Transacciones**

Se deberá crear al menos una transacción para ejecutar un Module Pool desde la opción del menú principal de SAP 'Herramientas→ Workbench ABAP →Desarrollo→Más herramientas→Transacciones' (SE93).

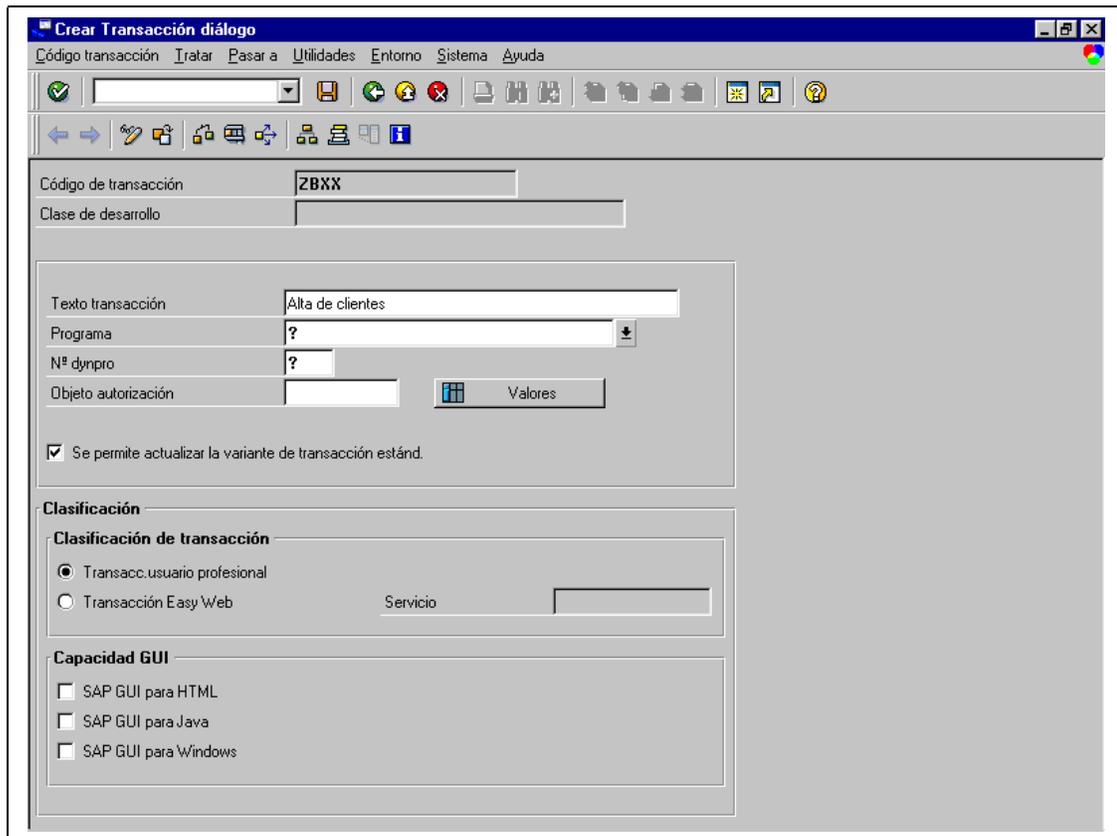
*Ej.: Se crea la transacción 'ZBXX' que ejecutará la pantalla de altas de clientes.*



Se informará el código de la transacción y, al pulsar el icono de crear , se presentará una pantalla en la que se seleccionará el tipo de transacción 'Transacción diálogo'.



Se introducen los datos de la transacción en la siguiente pantalla.



**Texto transacción:** Definición funcional de la transacción.

*Ej.: Alta de clientes.*

**Programa:** Nombre del programa marco del Module Pool que ejecutará la transacción.

*Ej.: 'SAPMZBXX'.*

**Nº dynpro:** Número de la pantalla inicial del Module Pool que ejecutará la transacción.

*Ej.: '9000'.*

**Objeto autorización:** Permite asignar un objeto de autorización a la transacción.

Se finalizará pulsando el icono de grabar  y seleccionando la opción de objeto local.

*Ej.: Ejecutar la pantalla de alta de clientes ejecutando la transacción 'ZBXX' desde la línea de comandos del sistema.*

### 5.2.7 Modificación dinámica de una pantalla

La definición gráfica de una pantalla se puede modificar dinámicamente durante su ejecución utilizando la tabla SCREEN, una tabla interna del sistema en la que se almacenan automáticamente los atributos de los campos de una pantalla en tiempo de ejecución.

Los atributos de un campo de pantalla se pueden modificar directamente sobre el registro correspondiente de la tabla SCREEN en el evento PBO.

Los campos que componen la tabla son los siguientes:

- SCREEN-NAME: Nombre del campo.
- SCREEN-GROUP1: Grupo de modificación 1.
- SCREEN-GROUP2: Grupo de modificación 2.
- SCREEN-GROUP3: Grupo de modificación 3.
- SCREEN-GROUP4: Grupo de modificación 4.
- SCREEN-REQUIRED: Atributo de campo obligatorio.
- SCREEN-INPUT: Atributo de campo de entrada.
- SCREEN-OUTPUT: Atributo de campo de salida.
- SCREEN-INTENSIFIED: Atributo de campo resaltado.
- SCREEN-INVISIBLE: Atributo de campo invisible.
- SCREEN-LENGTH: Longitud del campo.
- SCREEN-ACTIVE: Este campo está reservado para uso interno del sistema.
- SCREEN-DISPLAY\_3D
- SCREEN-VALUE\_HELP
- SCREEN-REQUEST

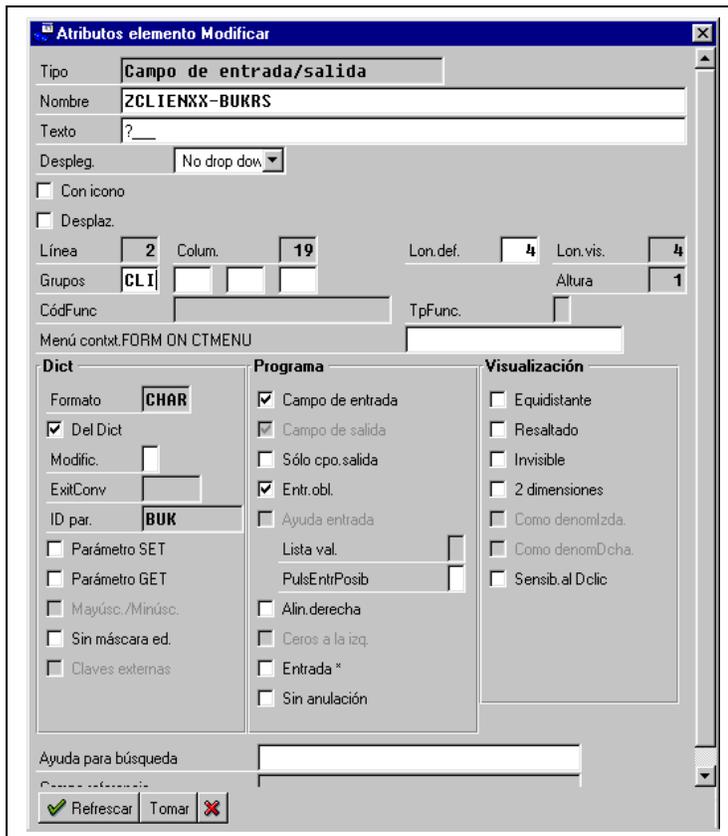
Para activar o desactivar atributos se asignará el valor '0' (desactivado) o '1' (activado).

Se pueden realizar modificaciones conjuntas de varios campos agrupándolos con el mismo valor en uno de los cuatro atributos de grupo.

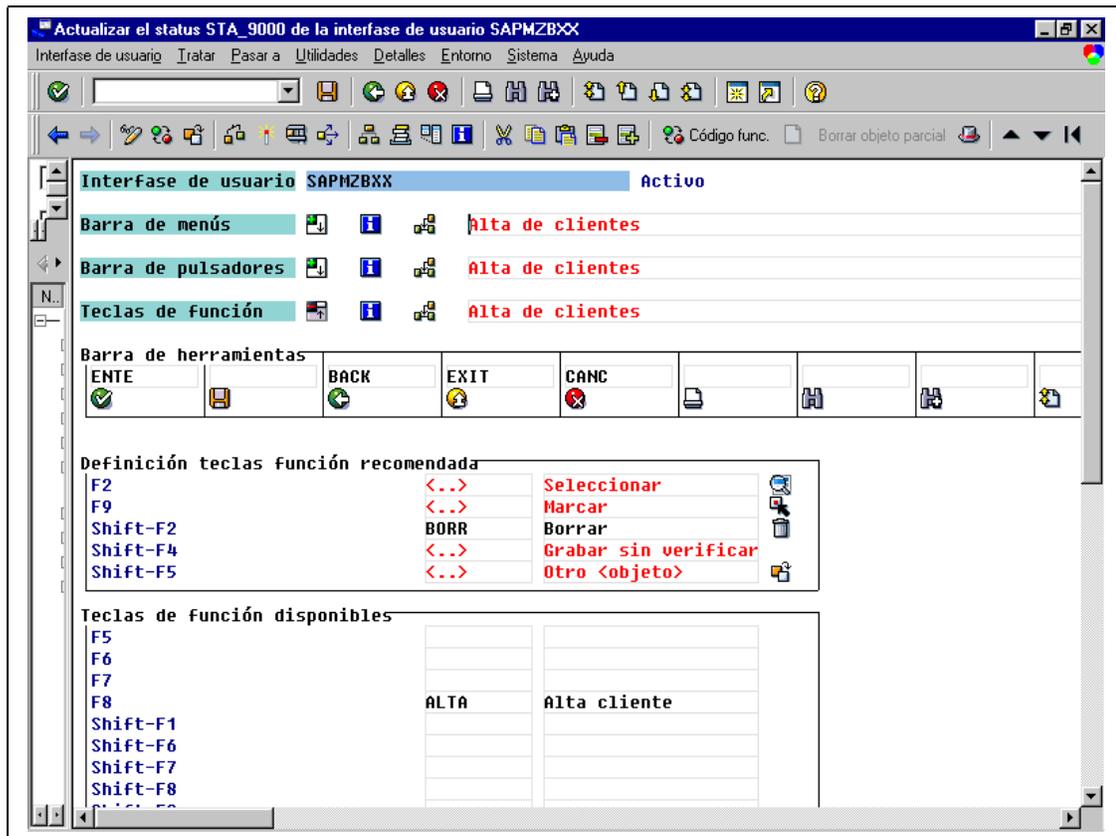
*EJ.: Se modificará la pantalla de altas de clientes evitando que se muestre el botón de altas hasta que no se chequeen los datos de pantalla pulsando ENTER, y una vez chequeados no se podrán modificar.*

*Se marcará el atributo 'Invisible' del campo de pantalla D\_ALTA en el editor gráfico SCREEN PAINTER.*

*Se informará el primer campo del atributo 'Grupos' con el literal 'CLI' en el editor gráfico SCREEN PAINTER en los siguientes campos: 'ZCLIENXX-BUKRS', 'ZCLIENXX-NCLIE', 'ZCLIENXX-NOMBR', 'ZCLIENXX-APEL1', 'ZCLIENXX-APEL2', 'ZCLIENXX-FNACI', 'D\_PARTICULAR', 'D\_EMPRESA' y 'D\_AUTOMATICO'. En las casillas y botones de selección hay que tener cuidado en escoger el campo de entrada de longitud 1 y no el literal descriptivo.*



Se asignará el código de función 'ENTE' con el texto 'Chequear datos' al símbolo enter  del STA\_9000 en el Menu Painter.



Se definirá la variable global `D_CLIENTE_CORRECTO` de tipo carácter y longitud 1.

Include `MZBXXTOP`:

\*\*\*\*\*

\* *Definición de variables*

\*\*\*\*\*

```
DATA: D_OKCODE LIKE SY-UCOMM, " Código de función de pantalla
 D_PARTICULAR(1) TYPE C, " Flag de cliente particular
 D_EMPRESA(1) TYPE C, " Flag de empresa
 D_AUTOMATICO(1) TYPE C, " Flag de numeración automática
 D_CLIENTE_CORRECTO(1) type c. " Flag de cliente correcto
```

Se incluye el tratamiento del código de función 'ENTE' en el módulo PAI 'USER\_COMMAND\_9000' activando el flag de cliente correcto, ya que si se ha llegado a ejecutar este módulo significa que no se ha lanzado ningún mensaje de error en los módulos de verificación previos y los datos chequeados son correctos.

`CASE D_OKCODE.`

```
 WHEN 'ALTA'. " Alta de cliente
 PERFORM ALTA_CLIENTE.
 WHEN 'BORR'. " Borrado de datos de cliente
 PERFORM BORRAR_DATOS.
```

```
WHEN 'ENTE'. " Chequear datos de cliente
 D_CLIENTE_CORRECTO = 'X'.
ENDCASE.
```

Se modifica la subrutina 'BORRAR\_DATOS' para que inicialice también el contenido del flag de cliente correcto.

```
* Se borran los datos del cliente.
CLEAR: ZCLIENXX,
 D_CLIENTE_CORRECTO.
```

Se modifica la subrutina 'ALTA\_CLIENTE' para que borre los datos del cliente después de realizar un alta correcta realizando una llamada a la subrutina 'BORRAR\_DATOS'.

```
* El cliente ha sido dado de alta
 MESSAGE S000(38) WITH TEXT-002.
* Se inicializan los datos del cliente después del alta
 PERFORM BORRAR_DATOS.
ENDIF.
```

Se creará el módulo PBO 'MODIFICAR\_PANTALLA\_9000' llamado al inicio del evento PBO, que desactivará la entrada de los campos del cliente y mostrará el botón de altas si está marcado el flag de cliente correcto, y hará lo contrario si no está marcado.

```
Lógica de proceso:
PROCESS BEFORE OUTPUT.
* Modificación de atributos de pantalla
 MODULE MODIFICAR_PANTALLA_9000.
```

```
Include MZBXX001:
&-----
*& Module MODIFICAR_PANTALLA_9000 OUTPUT
&-----
* Modifica los atributos de los campos de la pantalla de altas

MODULE MODIFICAR_PANTALLA_9000 OUTPUT.
```

```
LOOP AT SCREEN.
```

```
 IF SCREEN-NAME = 'D_ALTA'.
* Solamente se visualizará el botón de altas si está marcado el flag
* de cliente correcto.
 IF D_CLIENTE_CORRECTO IS INITIAL.
 SCREEN-INVISIBLE = 1.
 ELSE.
 SCREEN-INVISIBLE = 0.
```

```

ENDIF.
ENDIF.

```

```

IF SCREEN-GROUP1 = 'CLI'.

```

- \* Los campos de entrada agrupados con el literal 'CLI' en el grupo 1
- \* solo permitirán entradas si está desmarcado el flag de cliente
- \* correcto

```

IF D_CLIENTE_CORRECTO IS INITIAL.
 SCREEN-INPUT = 1.
ELSE.
 SCREEN-INPUT = 0.
ENDIF.

```

```

ENDIF.

```

- \* Se actualizan las modificaciones en la tabla SCREEN
- ```

MODIFY SCREEN.

```

```

ENDLOOP.

```

```

ENDMODULE.          " MODIFICAR_PANTALLA_9000 OUTPUT

```

5.2.8 Table Control

En una pantalla se pueden editar datos en una tabla utilizando el objeto TABLE CONTROL.

Ej.: Se crea la pantalla '9100' en el programa SAPMZBXX con el texto breve 'Visualización de clientes', tipo normal y dynpro siguiente '9100', que editará los clientes dados de alta en la tabla ZCLIENXX en un TABLE CONTROL.

Se asigna la variable 'D_OKCODE' en la lista de campos del editor gráfico, al campo de tipo OK que contiene el código de función seleccionado en la pantalla.

Se crea el Status 'STA_9100' con el texto 'Visualización de clientes' y tipo 'Dynpro' con el código de función 'EXIT' en el símbolo de finalizar , y el título '002' con el texto 'Visualización de clientes'.

Se crea el módulo PBO 'STATUS_9100' en el que se asigna el Status 'STA_9100' y el título '002' a la pantalla.

```

*&-----*
*&  Module STATUS_9100 OUTPUT
*&-----*
* Establece el Status y el título de la pantalla de visualización de
* clientes.
*&-----*

```

```
MODULE STATUS_9100 OUTPUT.
  SET PF-STATUS 'STA_9100'.
  SET TITLEBAR '002'.
```

```
ENDMODULE.          " STATUS_9100 OUTPUT
```

Se crea el módulo PAI 'USER_COMMAND_9100' que ejecutará las sentencias asociadas a la función seleccionada en la pantalla.

```
*&-----*
*&  Module USER_COMMAND_9100 INPUT
*&-----*
* Ejecuta las sentencias asociadas al código de función devuelto por
* la pantalla de visualización de clientes.
*-----*
```

```
MODULE USER_COMMAND_9100 INPUT.
```

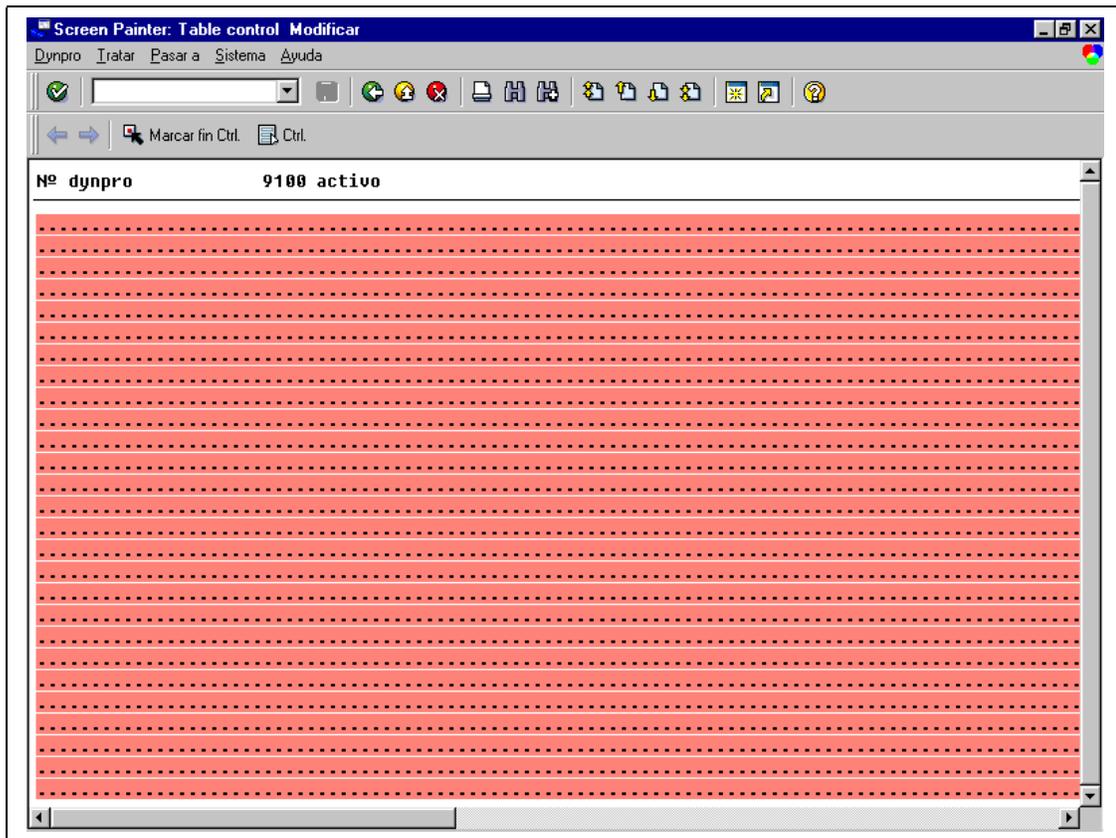
```
  CASE D_OKCODE.
    WHEN 'EXIT'. " Finalizar
    * Se finaliza la ejecución del programa
    LEAVE PROGRAM.
  ENDCASE.
```

```
ENDMODULE.          " USER_COMMAND_9100 INPUT
```

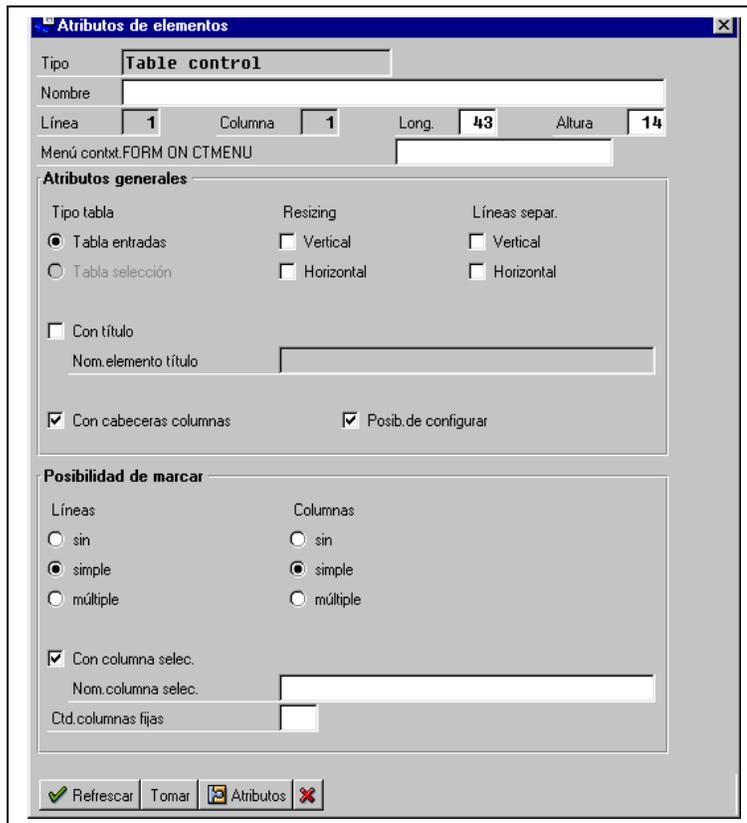
Se define la tabla interna I_CLIENTES en el Include de definiciones globales del programa, con la estructura de la tabla ZCLIENXX y un campo caracter de longitud 1, para almacenar los clientes que se van a visualizar en el TABLE CONTROL.

```
*****
* Definición de tablas internas
*****
* Tabla interna de clientes
DATA: BEGIN OF I_CLIENTES OCCURS 0.
      INCLUDE STRUCTURE ZCLIENXX.
DATA: MARCA,
      END OF I_CLIENTES.
```

Para crear un TABLE CONTROL ejecutaremos la opción de menú 'Tratar→Crear elemento→Table control' desde el editor gráfico del SCREEN PAINTER.



Se deberá especificar la posición final del TABLE CONTROL pulsando el botón  o haciendo doble click. Se informarán sus atributos en la ventana que aparece a continuación.



Nombre: Nombre de la variable asociada al TABLE CONTROL. Esta variable será un control de tipo TABLEVIEW.

Ej.: 'TC_CLIENTES'.

Atributos generales:

- Líneas separ.: Permite incluir líneas de separación horizontales y verticales entre los campos del TABLE CONTROL.
Ej.: Se activan las opciones 'Vertical' y 'Horizontal'.
- Con Tít.: Permite añadir un título al TABLE CONTROL en la línea superior, asociándole una variable en la que se informará el literal.
- Con cabeceras columnas: Permite añadir una línea de cabecera con las descripciones de las columnas.
Ej.: Marcado.
- Posib. De configurar: Permite la posibilidad de configurar la secuencia y el ancho de las columnas en tiempo de ejecución utilizando el icono  que aparece en el TABLE CONTROL.

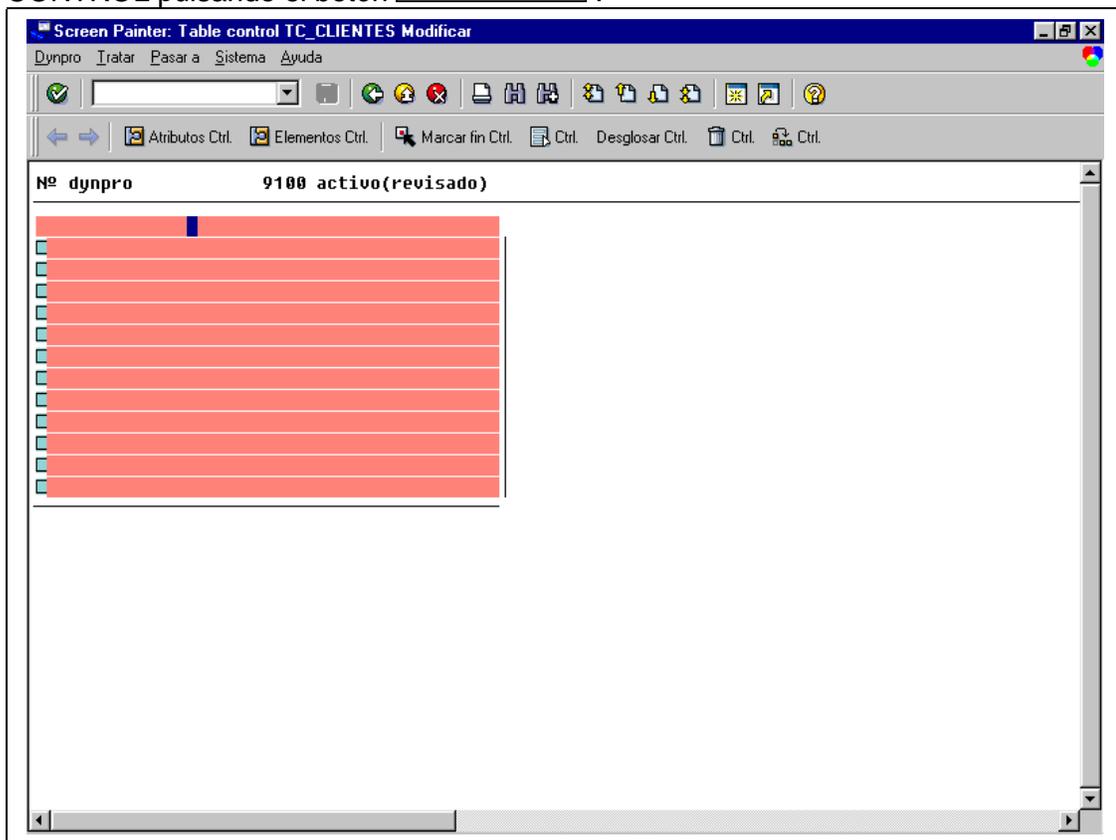
Posibilidad de marcar:

- Líneas: Especifica el número de líneas que se pueden marcar al mismo tiempo.
Ej.: 'simple'.
- Columnas: Especifica el número de columnas que se pueden marcar al mismo tiempo.
Ej.: 'sin'.

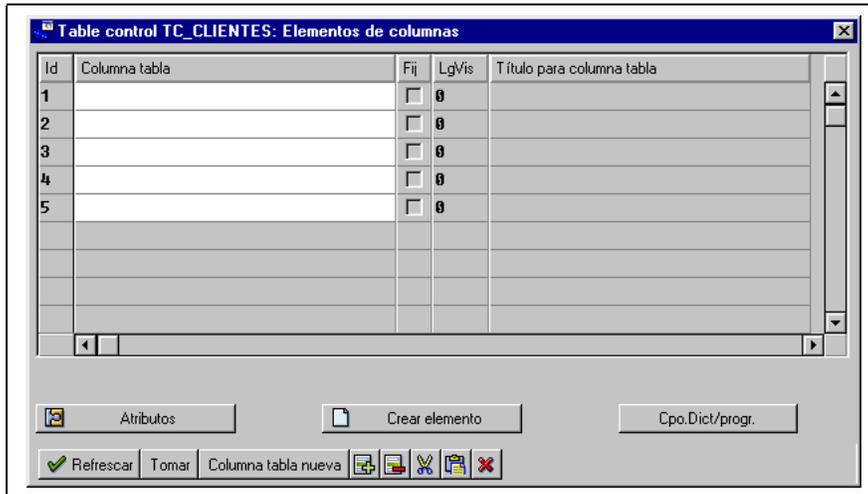
- Con columna marca: Al activar esta opción se incluye una columna de marcadores para poder seleccionar líneas, al inicio del TABLE CONTROL. Se deberá especificar un campo asociado que deberá estar definido como caracter de longitud 1. Esta opción es incompatible con la selección 'Sin posibilidad de marcar líneas'.
Ej.: marcado y asociado al campo I_CLIENTES-MARCA.
- Ctd. Columnas fijas: Especifica el número de columnas, comenzando por la izquierda, que permanecerán fijas en el TABLE CONTROL. Las columnas fijas no se podrán mover con el ratón ni se desplazarán al paginar horizontalmente.

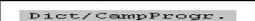
A continuación se deberán crear los campos que van a componer el TABLE

CONTROL pulsando el botón



En la siguiente pantalla , tendremos que indicar el nombre de los campos que lo componen.



Se pueden insertar campos asociados a variables ya definidas en el programa o campos del diccionario de datos seleccionándolos con el botón  y creándolos con el botón de pegar . Se pueden especificar los títulos de las columnas en las que aparecerán los campos pulsando el botón  sobre el campo 'Título para columna tabla' correspondiente, seleccionando el tipo de campo 'Texto/Palabra clave' e informando el literal en el atributo 'Txt. Campo'.

Ej.: Se insertan todos los campos de la tabla interna I_CLIENTES menos el mandante (I_CLIENTES-MANDT) y el flag de cliente marcado (I_CLIENTES_MARCA) utilizando el botón .

Se informan los títulos de los campos con los literales 'Soc.', 'Número', 'Nombre', 'Primer apellido', 'Segundo apellido' y 'Fecha nac.'



Con el botón  se pueden modificar los atributos de los campos que componen el TABLE CONTROL, posicionando el cursor en el campo correspondiente antes de pulsarlo.

Ej.: Se desmarca el atributo 'Campo entrada' en todos los campos del TABLE CONTROL para evitar que se pueda modificar su contenido.

Se finaliza la inserción de campos del TABLE CONTROL pulsando el botón .

Una vez terminada la definición gráfica se define el control asociado al TABLE CONTROL en el include de definiciones globales del programa utilizando la sentencia CONTROL <nombre> TYPE TABLEVIEW.

Ej.: Se define el control 'TC_CLIENTES' en el Include 'MZBXXTOP' con la siguiente sentencia:

** Definición de controles*

CONTROLS: TC_CLIENTES TYPE TABLEVIEW USING SCREEN 9100.

El control definido tendrá la siguiente campos:

- FIXED_COLS: Número de columnas fijas.
- LINES: Número de líneas que contienen datos.
- TOP_LINE: Primera línea visualizada en pantalla.
- CURRENT_LINE: Línea actual (dentro de un bucle LOOP ... ENDLOOP).
- LEFT_COL: Primera columna desplazable.
- LINE_SEL_MODE: Indicador de selección de líneas (0 = sin, 1 = simple, 2 = múltiple).
- COL_SEL_MODE: Indicador de selección de columnas (0 = sin, 1 = simple, 2 = múltiple).
- LINE_SELECTOR: Flag de utilización de columna marca.
- V_SCROLL: Flag de barra de scroll vertical.
- H_GRID: Flag de líneas de separación horizontal.
- V_GRID: Flag de líneas de separación vertical.
- COLS: Campo de tipo CXTAB_COLUMN que contiene un registro por cada columna del TABLE CONTROL con los siguientes campos:
- SCREEN: Atributos de pantalla con la misma estructura que la tabla interna del sistema SCREEN.
- INDEX: Posición de la columna.
- SELECTED: Flag de columna marcada.
- VISLENGTH: Ancho visible de la columna.
- INVISIBLE: Indicador de columna invisible.

En la lógica de proceso se debe tratar el TABLE CONTROL en los eventos PBO y PAI con las instrucciones LOOP y ENDLOOP. El tratamiento se puede realizar de dos formas:

- Con tabla interna.

Los campos que componen el TABLE CONTROL hacen referencia a los campos de una tabla interna que contiene los datos que se van a editar.

En el evento PBO se carga el TABLE CONTROL con las instrucciones LOOP y ENDLOOP de la siguiente forma:

```
LOOP AT <tabla_interna> WITH CONTROL <control>
      CURSOR <índice>.
      ...
ENDLOOP.
```

Cada vuelta del bucle LOOP se corresponde con una línea del TABLE CONTROL que se informa automáticamente con los valores contenidos en el registro de la tabla interna.

El índice especificado en la cláusula CURSOR se actualiza automáticamente con el número de línea del TABLE CONTROL que se está tratando. Se puede utilizar como índice el campo CURRENT_LINE del TABLE CONTROL.

Ej.: Se crea el módulo PBO 'INFORMAR_CLIENTES', llamado después del módulo 'STATUS_9100', que cargará la tabla interna de clientes con los datos de la tabla ZCLIENXX.

```
*-----*
*&  Module INFORMAR_CLIENTES OUTPUT
*&-----*
* Carga los registro de clientes de la tabla ZCLIENXX en la tabla
* interna de clientes.
*-----*
MODULE INFORMAR_CLIENTES OUTPUT.
DATA: L_LINEAS TYPE I. " Contador de registros de clientes

* Se comprueba si la tabla interna de clientes tiene registros
  DESCRIBE TABLE I_CLIENTES LINES L_LINEAS.

* Si está vacía se carga con los datos de la tabla de clientes
  IF L_LINEAS IS INITIAL.
    SELECT * INTO TABLE I_CLIENTES
      FROM ZCLIENXX.

* Se ordenan los registros
    SORT I_CLIENTES.

  ENDIF.

ENDMODULE.          " INFORMAR_CLIENTES OUTPUT
```

Se cargarán los datos de la tabla interna I_CLIENTES en el TABLE CONTROL TC_CLIENTES en el evento PBO de la lógica de proceso después de la llamada al módulo 'INFORMAR_CLIENTES'.

PROCESS BEFORE OUTPUT.

* *Establecimiento de STATUS de pantalla*

MODULE STATUS_9100.

* *Carga de datos en la tabla interna de clientes*

MODULE INFORMAR_CLIENTES.

* *Se cargan los datos de clientes en el TABLE CONTROL*

LOOP AT I_CLIENTES

 WITH CONTROL TC_CLIENTES

 CURSOR TC_CLIENTES-CURRENT_LINE.

ENDLOOP.

En el evento PAI se leen las líneas del TABLE CONTROL con las instrucciones LOOP y ENDLOOP de la siguiente forma:

```
LOOP AT <tabla_interna>.
```

```
    ...
```

```
    MODULE <modulo>.
```

```
    ...
```

```
ENDLOOP.
```

En cada vuelta del bucle LOOP se informa el contenido de una línea del TABLE CONTROL en la cabecera de la tabla interna.

Ej.: Se crean las sentencias LOOP y ENDLOOP en el evento PAI antes de ejecutar el módulo 'USER_COMMAND_9100', aunque no se realiza ningún tratamiento con los datos del TABLE CONTROL.

PROCESS AFTER INPUT.

* *Se accede a los datos del TABLE CONTROL*

LOOP AT I_CLIENTES.

ENDLOOP.

* *Control de funciones de usuario*

MODULE USER_COMMAND_9100.

- Sin tabla interna.

En el evento PBO se carga el TABLE CONTROL con las instrucciones LOOP y ENDLOOP de la siguiente forma:

```
LOOP WITH CONTROL <control>.
```

```
    ...
```

```
    MODULE <modulo>.
```

```
    ...
```

```
ENDLOOP.
```

Cada vuelta del bucle LOOP se corresponde con una línea del TABLE CONTROL, debiendo asignar el contenido a sus campos en un módulo. Los campos de un TABLE CONTROL se referencian igual que los de una tabla interna (control-campo). Se puede finalizar la introducción de líneas en el TABLE CONTROL con la sentencia EXIT FROM STEP-LOOP, que provoca la salida del bucle LOOP.

En el evento PAI se leen las líneas del TABLE CONTROL con las instrucciones LOOP y ENDLOOP de la siguiente forma:

```
LOOP WITH CONTROL <control>.  
  ...  
  MODULE <módulo>.  
  ...  
ENDLOOP.
```

En cada vuelta del bucle LOOP se puede acceder al contenido de una línea del TABLE CONTROL en un módulo.

Es obligatorio especificar las sentencias LOOP y ENDLOOP en los eventos PBO y PAI siempre que exista un TABLE CONTROL en la pantalla.

Dentro de un bucle LOOP ... ENDLOOP la variable del sistema SY-LOOPC contiene el número de líneas table control que se visualizan simultáneamente en la pantalla, y la variable SY-STEPL contiene el número de línea que se está tratando.

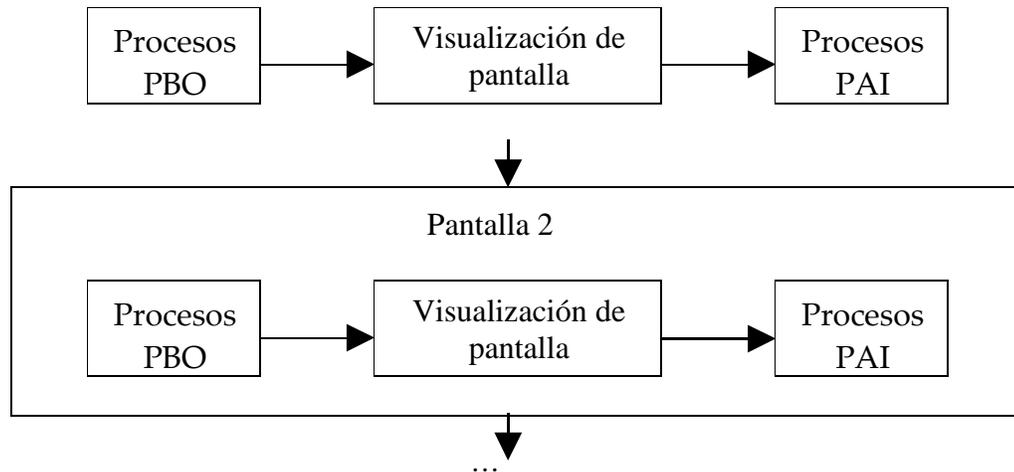
Se puede determinar la línea del TABLE CONTROL en la que está posicionado el cursor con la instrucción GET CURSOR LINE <variable>.

Ej.: Se crea la transacción 'ZCXX', que ejecutará la pantalla de visualización de clientes, con el tipo 'Transacción de diálogo', el texto 'Visualización de clientes', el programa 'SAPMZBXX' y la pantalla '9100'.

Se ejecuta la pantalla de visualización de clientes ejecutando la transacción 'ZCXX' desde la línea de comandos del sistema

5.2.9 Secuencia de proceso de pantallas

Después de procesarse el evento PAI de una pantalla se ejecutará la pantalla especificada en el atributo 'Pantalla siguiente'.



Se puede modificar dinámicamente la secuencia de ejecución de las pantallas en el programa utilizando las siguientes sentencias:

- SET SCREEN.

La sentencia 'SET SCREEN <pantalla>' modifica temporalmente el valor del atributo 'Dynpro siguiente'. La siguiente vez que se ejecute la pantalla el atributo tendrá su valor original.

- LEAVE SCREEN.

Esta sentencia finaliza la ejecución de la pantalla y provoca la ejecución de la pantalla especificada en el atributo 'Dynpro siguiente'.

- LEAVE TO SCREEN.

La sentencia 'LEAVE TO SCREEN <pantalla>' finaliza la ejecución de la pantalla y provoca la ejecución de la pantalla especificada sin tener en cuenta el atributo 'Dynpro siguiente'.

- CALL SCREEN.

La sentencia 'CALL SCREEN <pantalla>' ejecuta la pantalla especificada. Al finalizar la ejecución de la pantalla especificada (con la sentencia 'LEAVE TO SCREEN 0' o si finaliza su ejecución y no tiene informado el atributo 'Dynpro siguiente') se continuará ejecutando la pantalla que realizó la llamada en la siguiente instrucción a la sentencia de llamada.

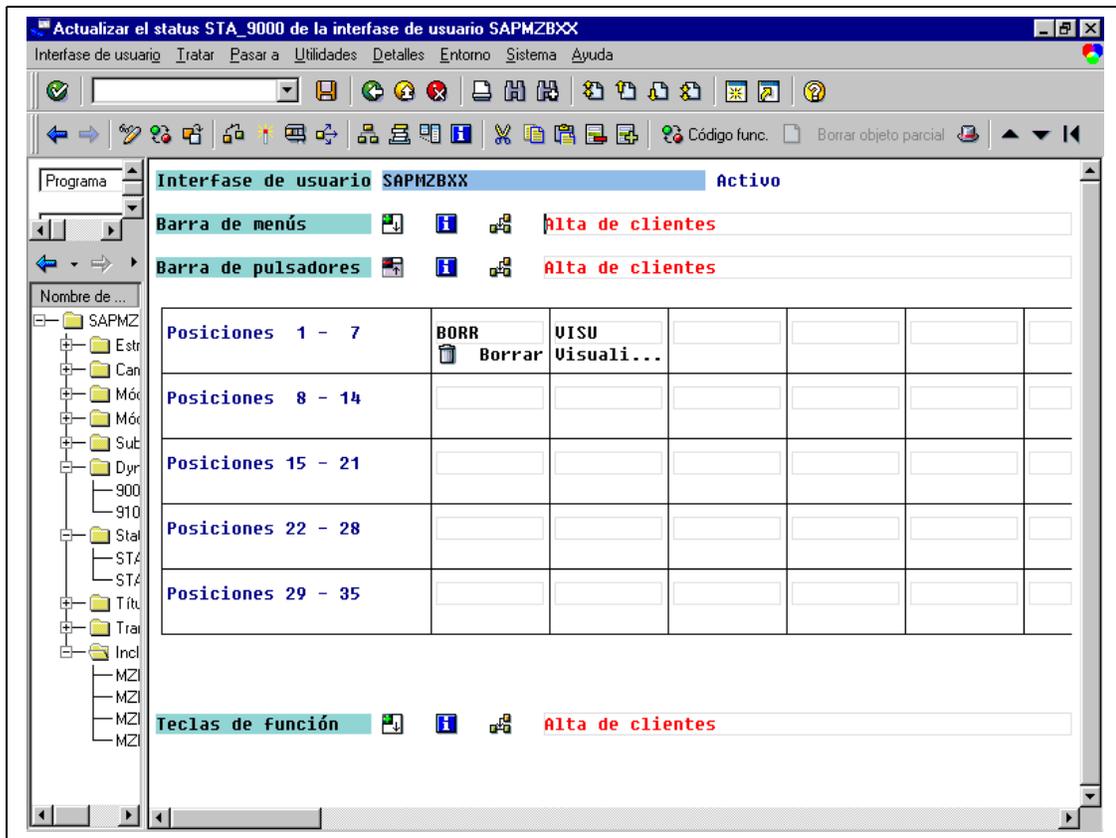
Con las cláusulas 'STARTING AT <columna> <fila>' y 'ENDING AT <columna> <fila>' se puede especificar el tamaño de la pantalla indicando las coordenadas iniciales y finales respectivamente. Al realizar una llamada a una pantalla con estas cláusulas, el sistema asumirá que la pantalla llamada es de diálogo modal.

- LEAVE PROGRAM.

Esta sentencia finaliza inmediatamente la ejecución del programa.

Ej.: Se incluirá un pulsador en la pantalla de altas de clientes para ejecutar la pantalla de visualización de clientes.

Se incluye un pulsador en la barra de pulsadores el Status 'STA_9000' con el código de función 'VISU' de tipo 'E' (Comando Exit) y el texto 'Visualizar clientes' asociado a la tecla de función 'F5'.



Se modifica el establecimiento del Status de la pantalla de altas en el módulo 'STATUS_9000' para que no muestre el botón de Visualización de clientes si está marcado el flag de cliente correcto.

MODULE STATUS_9000 OUTPUT.

```

* Solamente aparecerá el botón de visualización de clientes cuando no se
* hayan introducido datos correctos de un cliente
IF D_CLIENTE_CORRECTO IS INITIAL.
  SET PF-STATUS 'STA_9000'.
ELSE.
  SET PF-STATUS 'STA_9000' EXCLUDING 'VISU'.
ENDIF.

```

SET TITLEBAR '001'.

ENDMODULE. " STATUS_9000 OUTPUT

Se añade el procesamiento del código de función 'VISU' en el módulo PAI 'SALIR_9000', realizando una llamada a la pantalla '9100' en caso de que se selecciones la función 'Visualizar clientes'.

WHEN 'VISU'. " Visualizar clientes
** Se ejecuta la pantalla de visualización de clientes*
CALL SCREEN '9100'.

Se activa el símbolo de regresar en el Status 'STA_9100' con el código de función 'BACK'.

Se añade el procesamiento del código de función 'BACK' en el módulo PAI 'USER_COMMAND_9100', regresando a la pantalla anterior en caso de que se selecciones la función 'Regresar'.

WHEN 'BACK'. " Regresar
** Se regresa a la pantalla anterior*
LEAVE TO SCREEN 0.

Se ejecuta la pantalla de visualización de clientes desde la pantalla de altas de clientes pulsando el botón .

5.2.10 Procesamiento de listados en pantallas

Se puede procesar un listado desde una pantalla utilizando la sentencia LEAVE TO LIST PROCESSING, pudiendo utilizar desde ese momento las sentencias y eventos ABAP/4 asociados a los listados. Se ejecutarán todos los módulos PAI de la pantalla antes de que se visualice el listado.

Se puede establecer un Status creado a medida para el listado o asignar el Status estándar para listados con la sentencia SET PF-STATUS SPACE.

Al finalizar el procesamiento del listado, al pulsar el símbolo de regresar en el listado o mediante la sentencia LEAVE LIST-PROCESSING en el programa, se continúa ejecutando la siguiente pantalla. Con la cláusula AND RETURN TO SCREEN de la sentencia LEAVE TO LIST-PROCESSING se puede especificar la pantalla que se ejecutará al finalizar el listado.

Ej.: Se mostrará un listado con las facturas del cliente que tenga marcada la casilla de selección del TABLE CONTROL de la pantalla de visualización de clientes al pulsar ENTER.

Se define la tabla de facturas ZCLIENXX en el Include MZBXXTOP.

** Definición de tablas*

```
TABLES: ZCLIENXX,      " Maestro de clientes
        T001,         " Sociedades
        ZFACTUXX.     " Facturas de clientes
```

Se define las variables D_BUKRS_SEL y D_NCLI_SEL en el Include MZBXXTOP con la estructura de los campos ZCLIENXX-BUKRS y ZCLIENXX-NCLIE respectivamente para almacenar la sociedad y el número de cliente del cliente seleccionado en el TABLE CONTROL de clientes.

** Definición de variables*

```
DATA: D_OKCODE LIKE SY-UCOMM,      " Código de función de pantalla
      D_PARTICULAR(1) TYPE C,      " Flag de cliente particular
      D_EMPRESA(1) TYPE C,         " Flag de empresa
      D_AUTOMATICO(1) TYPE C,      " Flag de numeración automática
      D_CLIENTE_CORRECTO(1) TYPE C, " Flag de cliente correcto
      D_BUKRS_SEL LIKE ZCLIENXX-BUKRS, " Sociedad de cliente selec.
      D_NCLIE_SEL LIKE ZCLIENXX-NCLIE. " Cliente seleccionado
```

Se crea el módulo 'CLIENTE_SELECCIONADO' asociado al campo de pantalla 'I_CLIENTES_MARCA' que se ejecutará dentro del bucle LOOP ... ENDLOOP del evento PAI de la pantalla de visualización de clientes, almacenando la sociedad y el número de cliente en las variables correspondientes cuando se trate una línea del table control en la que el campo MARCA este activo.

Lógica de proceso:

** Se accede a los datos del TABLE CONTROL*

```
LOOP AT I_CLIENTES.
  FIELD I_CLIENTES-MARCA
  MODULE CLIENTE_SELECCIONADO ON INPUT.
ENDLOOP.
```

Include MZBXXI01:

```
*&-----*
*&  Module CLIENTE_SELECCIONADO INPUT
*&-----*
* Informa la sociedad y el número del cliente seleccionado en el TABLE
* CONTROL de clientes en las variables correspondientes.
*-----*
```

MODULE CLIENTE_SELECCIONADO INPUT.

```
D_BUKRS_SEL = I_CLIENTES-BUKRS.  
D_NCLIE_SEL = I_CLIENTES-NCLIE.
```

```
ENDMODULE.          " CLIENTE_SELECCIONADO INPUT
```

Se añade el procesamiento del resto de códigos de función no tratados (en este caso se procesa la tecla ENTER) en el módulo PAI 'USER_COMMAND_9100' que ejecutará la subrutina 'LISTADO_FACTURAS'.

```
WHEN OTHERS.  
    PERFORM LISTADO_FACTURAS.  
ENDCASE.
```

Se crea la subrutina 'LISTADO_FACTURAS', que mostrará un listado con las facturas del cliente seleccionado en el TABLE CONTROL de clientes.

```
*&-----*  
*&   Form LISTADO_FACTURAS  
*&-----*  
* Visualiza un listado con las facturas del cliente seleccionado.  
*-----*
```

```
FORM LISTADO_FACTURAS.
```

```
* Se comprueba que se ha seleccionado un cliente  
CHECK NOT D_NCLIE_SEL IS INITIAL.  
* Se pasa al modo de proceso de listados  
LEAVE TO LIST-PROCESSING.  
* Se establece el Status estándar de listados  
SET PF-STATUS SPACE.  
* Se escriben las facturas correspondientes al cliente seleccionado  
SELECT * FROM ZFACTUXX  
    WHERE BUKRS = D_BUKRS_SEL  
    AND NCLIE = D_NCLIE_SEL.  
    WRITE: / ZFACTUXX-NFACT,  
ZFACTUXX-IMPNT CURRENCY ZFACTUXX-MONEDA,  
    ZFACTUXX-MONEDA.  
ENDSELECT.  
* Se comprueba si se ha escrito alguna factura  
IF SY-SUBRC <> 0.  
* Error: No existen facturas para el cliente seleccionado  
MESSAGE S000(38) WITH TEXT-010.  
ENDIF.  
* Se inicializan los datos del cliente seleccionado  
CLEAR: D_BUKRS_SEL,  
    D_NCLIE_SEL.
```

```
ENDFORM.          " LISTADO_FACTURAS
```

Se crea el elemento de texto TEXT-010 con el texto 'No existen facturas para el cliente seleccionado'.

Se crea el include MZBXXF02 en el programa marco SAPMZBXX con el texto 'Include de eventos de listado del programa SAPMZBXX', tipo 'I' y aplicación '**', que contendrá los eventos de listado del programa.

PROGRAM SAPMZBXX.

INCLUDE MZBXXTOP.

INCLUDE MZBXXO01.

INCLUDE MZBXXI01.

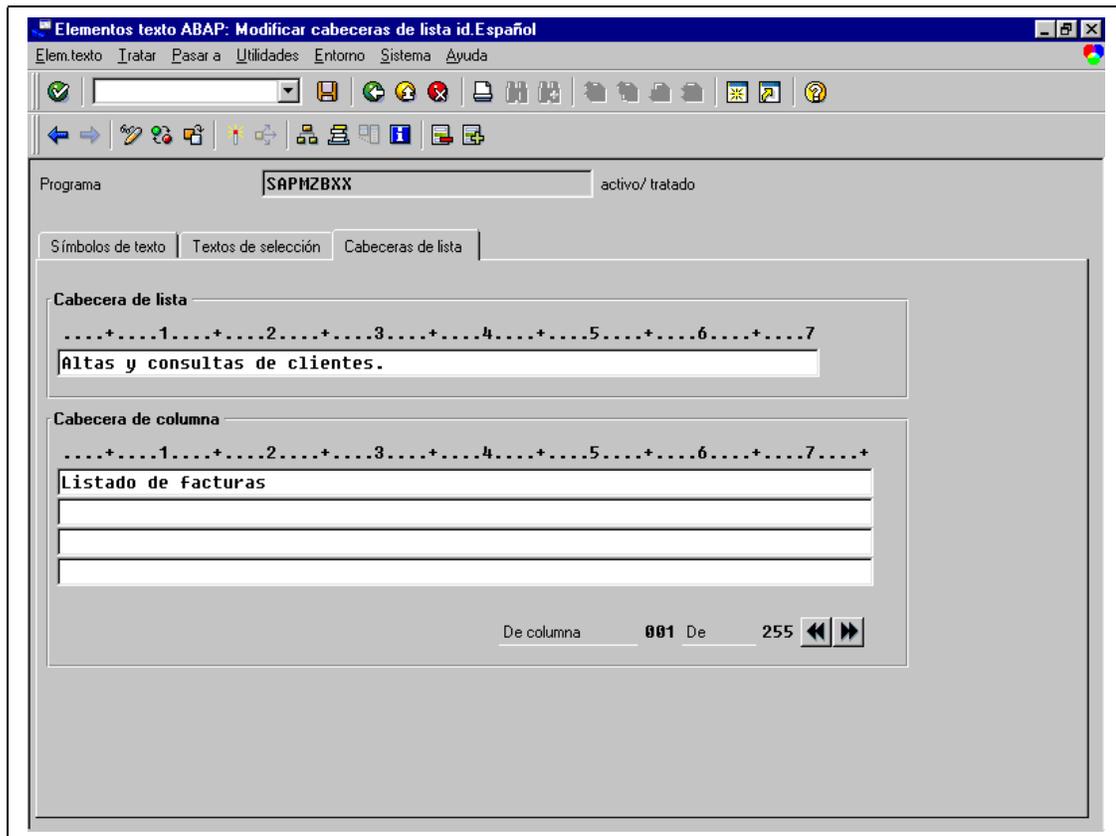
INCLUDE MZBXXF01.

INCLUDE MZBXXF02.

Se crea el evento TOP-OF-PAGE en el Include MZBXXF02, que escribirá las cabeceras del listado de facturas.

```
*-----*
* INCLUDE MZBXXF02 *
*-----*
TOP-OF-PAGE.
* Se escribe el número del cliente seleccionado
WRITE: 'Cliente:',
      D_NCLIE_SEL.
ULINE.
* Se escriben las cabeceras de columna
WRITE: (10) 'Factura',
      (16) 'Importe' RIGHT-JUSTIFIED,
      'Moneda'.
SKIP.
```

Se establece el literal 'Listado de facturas' como 'Cabecera de lista' en los elementos de texto del programa marco SAPMZBXX.



Se ejecuta el listado de facturas seleccionando un cliente en el TABLE CONTROL de la pantalla de visualización de clientes y pulsando la tecla ENTER.

Se puede procesar un listado en una ventana de diálogo llamando a la pantalla con las cláusulas STARTING AT y ENDING AT. La pantalla deberá ejecutar la sentencia SUPPRESS DIALOG en el evento PBO para evitar que se visualice, y la sentencia LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0 para iniciar el procesamiento del listado y volver a la pantalla que realizó la llamada al finalizar el mismo.

6. Módulos de Funciones

6.1 Introducción

Los módulos de funciones son objetos que realizan operaciones que pueden ser utilizadas en varios programas. Al crear un módulo de función con el código que realiza una operación, se evita tener que repetirlo en todos los programas que realicen esa operación añadiendo en ellos una llamada al módulo de función. Además de evitar que se repita el mismo código en diferentes programas, se facilita el mantenimiento del proceso, ya que las modificaciones que se realicen en un módulo de función afectan a todos los programas que lo utilicen.

SAP dispone de un gran número de módulos de función predefinidos que se pueden utilizar en nuestros programas, a las que se añadirán los que se desarrollen a medida.

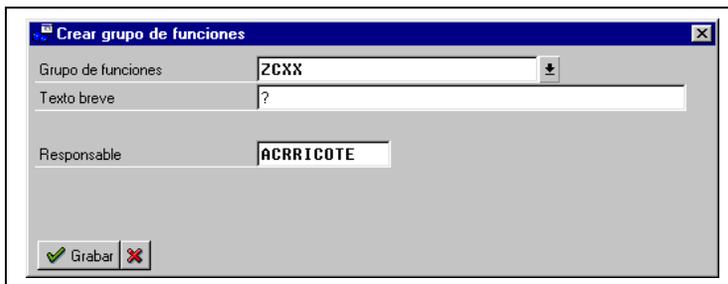
Los módulos de función pertenecen a grupos de funciones, que los agrupan según su funcionalidad. Los módulos de funciones de un mismo grupo de funciones comparten las definiciones de datos globales.

6.2 Creación de un grupo de funciones.

Para crear un grupo de función iremos por la opción de menú del OBJEKT NAVIGATOR
Ruta de acceso: (En el menú principal de SAP) 'Herramientas→ Workbench ABAP4→ Resumen → Object Navigator (SE80).

Seleccionaremos 'Grupo de funciones' , pondremos el nombre del grupo de funciones a Crear
Ej 'ZCXX'.

Aparecerá una ventana en la que definir los atributos del grupo de funciones.



Grupo de funciones: Nombre del grupo de funciones.

Texto breve: Descripción del grupo de funciones.

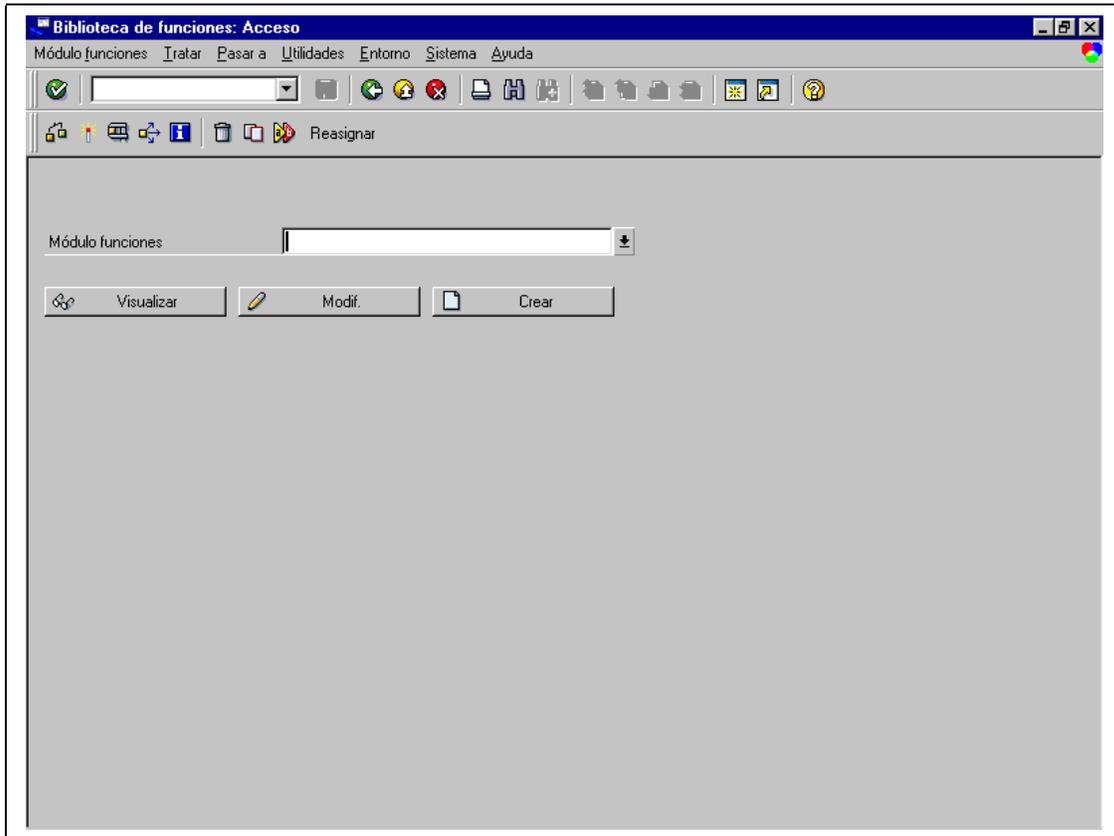
Ej: 'Asignación de números'.

Responsable: Usuario responsable del grupo de funciones.

6.3 Datos de gestión

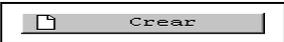
Los módulos de función se mantienen utilizando la biblioteca de funciones.

Ruta de acceso: (En el menú principal de SAP) 'Herramientas→ Workbench ABAP4-Desarrollo→Biblioteca funciones' (SE37).



Desde esta pantalla se pueden crear, modificar o visualizar todas las partes de un módulo de función marcando las distintas opciones de objetos parciales.

Ej.: Se creará una función que recibirá como parámetro de entrada una sociedad y devolverá como parámetro de salida el primer número de cliente desocupado en la tabla de clientes ZCLIENXX para esa sociedad.

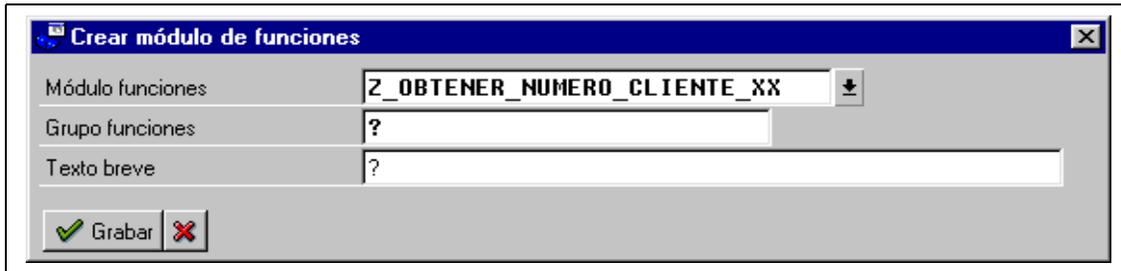
Para crear un módulo de función se deberá especificar el nombre en la pantalla inicial y pulsar el botón de crear  con la opción 'Gestión' activada.

Ej.: 'Z_OBTENER_NUMERO_CLIENTE_XX'.

Aparecerá una pantalla en la que se debe indicar el grupo de funciones a la que pertenece el módulo de función junto con su la descripción de la función.

Ej.: 'ZCXX'..

'Devuelve un número de cliente libre'.



Crear módulo de funciones

Módulo funciones: Z_OBTENER_NUMERO_CLIENTE_XX

Grupo funciones: ?

Texto breve: ?

Grabar

Una vez especificado el grupo de función se deberán informar los datos de gestión del módulo de función.

Clasificación:

- Aplicación: Módulo al que pertenece el programa (FI , HHRR ...).
Ej.: '' Multiaplicación.*
- Texto breve: Descripción de la funcionalidad del módulo de función.
Ej.: 'Determinación de número de cliente'.

Forma ejec.:

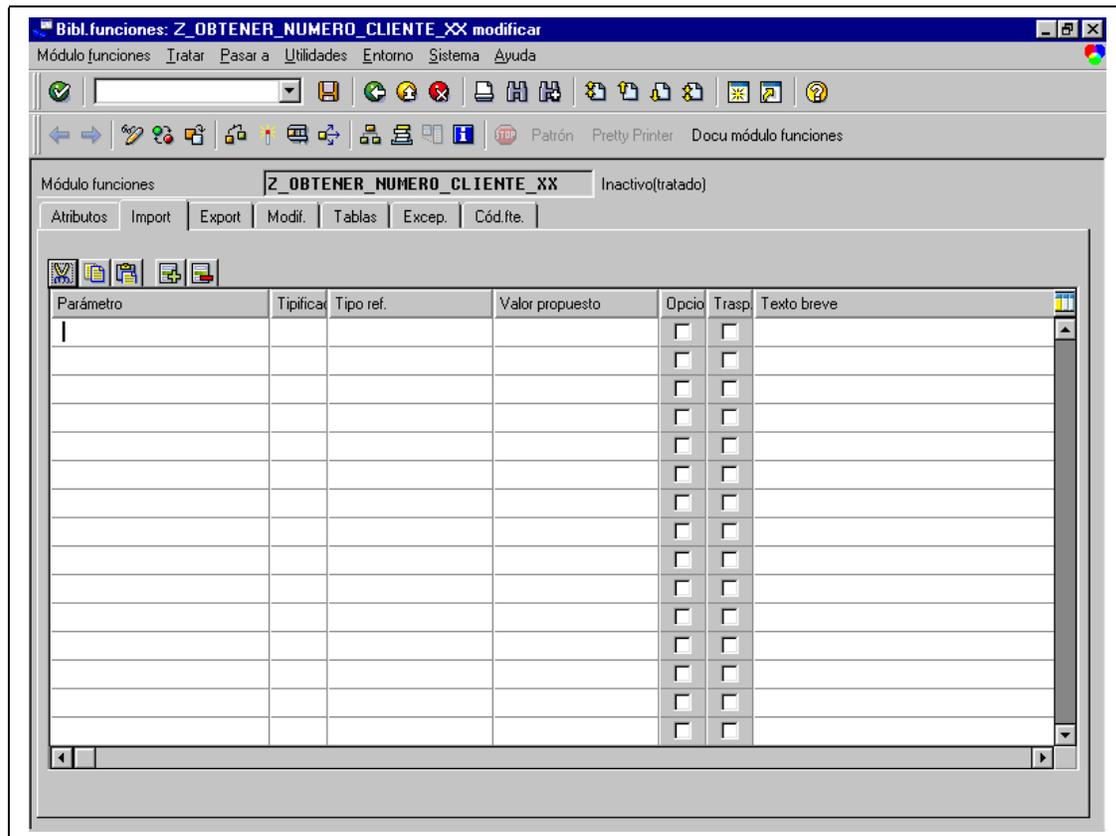
- Normal: Módulo de función normal.
- Apoyo Remote Function Call: Funciones de ejecución remota. Estas funciones pueden ser ejecutadas desde otros sistemas externos a SAP.
- Actualizable: Funciones de actualización asíncrona. Se puede especificar el modo de tratamiento de la tarea de actualización (Inicio inmediato, inicio inmediato sin actualización posterior, inicio retardado o lanzamiento colectivo).
Ej.: 'Normal'.

Al finalizar la introducción de los datos de gestión se deberá grabar el módulo de función.

6.4 Parámetros de la función.

6.4.1 Parámetros Import.

Deberemos indicar la lista de parámetros de entrada de la función:



Parámetro: Nombre de los parámetros de entrada.

Tipificación: Tipo de dato del parámetro.

Tipo Ref. Tipo de referencia (Solo se indicará una de estas dos últimas).

Valor Propuesto: Valor por defecto cuando se declara como opcional.

Flag opc. Indicador de parámetro no obligatorio.

Flag. Tras. Indicación de que el parámetro se recibe por valor/referencia.

6.4.2 Parámetros de salida (Export).

Son los parámetros de salida de función.

6.4.3 Parámetros (changing):

Parámetros de entrada y salida del módulo de función. Estos parámetros devuelven el valor de salida al finalizar la ejecución del módulo de función sin errores, manteniendo el valor de entrada en caso contrario.

Ej.: Se crea el parámetro import 'BUKRS' con el campo de referencia 'ZCLIENXX-BUKRS' para recibir la sociedad del cliente, y el parámetro export 'NCLIE' con el campo de referencia 'ZCLIENXX-NCLIE' para devolver el número de cliente.

de excepciones se corresponderá con el valor que devolverá en la variable SY-SUBRC (la primera excepción definida devolverá 1, la siguiente 2, etc.).

Por defecto siempre existe la excepción 'OTHERS', aunque aparezca definida, que se utiliza para devolver un error genérico y devuelve en la variable SY-SUBRC el valor siguiente al de la última excepción creada.

Ej.: Se crea la excepción 'SOCIEDAD_INEXISTENTE' que será devuelta cuando la sociedad informada en el parámetro de entrada correspondiente no exista en la tabla estándar de sociedades.

Para devolver una excepción desde el código del módulo de función se utiliza la sentencia RAISE <excepción>, finalizando así la ejecución de la función y devolviendo el código asociado a la excepción en la variable SY-SUBRC. En caso de devolver una excepción no se actualiza el valor de salida de los parámetros CHANGING.

Con la cláusula RAISING de la instrucción MESSAGE se puede dar la posibilidad de que el módulo de función trate el error mostrando el mensaje de error especificado o que devuelva la excepción correspondiente sin mostrar el mensaje de error, en función de si se especifica la cláusula 'EXCEPTIONS' en su llamada. Si no se especifica la cláusula, el módulo de función mostrará los mensajes de error que tengan la cláusula 'RAISING', finalizando así la ejecución del programa que realiza la llamada, en caso contrario se devolverán las excepciones asociadas a los mensajes traspasando el control de los errores al programa que realiza la llamada al módulo de función.

6.7 Datos globales

Las definiciones globales de datos son compartidas por todos los módulos de función de un grupo de funciones. Las definiciones de objetos globales se mantienen a través de la opción de menú 'Pasar a → Datos globales'.

Nota: Las definiciones globales y el texto fuente del módulo de función se codifican realmente en el editor ABAP/4 utilizando los mismos comandos que en la codificación de listados.

Ej.: Se define la tabla estándar de sociedades 'T001' en la que se chequeará que la sociedad recibida como parámetro es correcta, y la tabla de clientes 'ZCLIENXX' para seleccionar los números de cliente existentes.

```
Include LZCLITOP:
FUNCTION-POOL ZCLI.                "MESSAGE-ID ..
*****
* Definición de tablas
*****
TABLES: T001,    " Sociedades
        ZCLIENXX. " Clientes
```

Los parámetros definidos en el módulo de función son locales, por lo tanto solo son visibles en el cuerpo principal de la función. Para que puedan ser utilizados en las subrutinas, sin necesidad de pasarlos como parámetros, se deberán globalizar utilizando la opción de menú 'Tratar→Interfase→Globalizar parám.' en la pantalla de mantenimiento de parámetros de entrada/salida. Para eliminar la globalización se utiliza la opción de menú 'Tratar→Interfase→Localizar parám.'

6.8 Código fuente

Al crear una subrutina, haciendo doble click sobre el nombre en la llamada, el sistema propondrá la creación de un Include que contendrá todas las subrutinas del grupo de funciones.

Al finalizar la introducción del texto fuente se deberá grabar, verificar y activar el módulo de función.

Ej.: Se crea la subrutina 'CHEQUEAR_SOCIEDAD' en un nuevo Include LZCLIF01 pasándole como parámetro de entrada el parámetro de sociedad, que chequeará que la sociedad existe en la tabla estandar de sociedades, devolviendo la excepción 'SOCIEDAD_INEXISTENTE' en caso contrario.

Se crea la subrutina 'SELECCIONAR_NUMERO_CLIENTE' pasándole como parámetro de entrada el parámetro de sociedad, que devolverá como parámetro de salida el primer número de cliente desocupado para la sociedad indicada.

Include LZCLIU01:

FUNCTION Z_OBTENER_NUMERO_CLIENTE.

```

*-----
**Interfase local
**  IMPORTING
**    VALUE(BUKRS) LIKE ZCLIENXX-BUKRS
**    EXPORTING
**    VALUE(NCLIE) LIKE ZCLIENXX-NCLIE
**    EXCEPTIONS
**    SOCIEDAD_INEXISTENTE
*-----

```

** Se chequea la sociedad*

PERFORM CHEQUEAR_SOCIEDAD USING BUKRS.

** Se selecciona el número de cliente para la sociedad*

PERFORM SELECCIONAR_NUMERO_CLIENTE USING BUKRS NCLIE.

ENDFUNCTION.

Include LZCLIF01:

```

*-----

```

```
***INCLUDE LZCLIF01 .
```

```
*-----*
```

```
*&-----*
```

```
*& Form CHEQUEAR_SOCIEDAD
```

```
*&-----*
```

```
* Chequea que la sociedad recibida como parámetro existe en la tabla  
* de sociedades.
```

```
*-----*
```

```
* --> PE_BUKRS Sociedad
```

```
*-----*
```

```
FORM CHEQUEAR_SOCIEDAD USING VALUE(PE_BUKRS) LIKE ZCLIENXX-  
BUKRS.
```

```
* Se selecciona la sociedad en la tabla de sociedades
```

```
SELECT SINGLE BUKRS INTO T001-BUKRS
```

```
FROM T001
```

```
WHERE BUKRS = PE_BUKRS.
```

```
* Si no se ha seleccionado se devuelve la excepción SOCIEDAD_INEXISTENTE
```

```
IF SY-SUBRC <> 0.
```

```
RAISE SOCIEDAD_INEXISTENTE.
```

```
ENDIF.
```

```
ENDFORM. " CHEQUEAR_SOCIEDAD
```

```
*&-----*
```

```
*& Form SELECCIONAR_NUMERO_CLIENTE
```

```
*&-----*
```

```
* Devuelve en el parámetro de salida PS_CLIE el primer número de cliente
```

```
* desocupado en la tabla de clientes para la sociedad especificada en el
```

```
* parámetro de entrada PE_BUKRS.
```

```
*-----*
```

```
* -->PE_BUKRS Sociedad
```

```
* -->PS_NCLIE Número de cliente *
```

```
*-----*
```

```
FORM SELECCIONAR_NUMERO_CLIENTE  
USING VALUE(PE_BUKRS) LIKE ZCLIENXX-BUKRS  
PS_NCLIE LIKE ZCLIENXX-NCLIE.
```

```
* Se selecciona el nº mayor existente en la tabla
```

```
SELECT MAX( NCLIE )
```

```
FROM ZCLIENXX INTO (ZCLIENXX-NCLIE)
```

```
WHERE BUKRS = PE_BUKRS.
```

```
* Si se encuentra valor máximo
```

```
IF ( SY-SUBRC = 0 ).
```

```
PS_NCLIE = ZCLIENXX-NCLIE + 1.
```

```
* Si no se encuentra valor máximo
```

```
PS_NCLIE = 1.
```

ENDIF.

* *Completamos con ceros por la Izq.*

UNPACK PS_NCLIE.

ENDFORM. " SELECCIONAR_NUMERO_CLIENTE

6.9 Ejecución

Para ejecutar un módulo de función desde la biblioteca de funciones se utiliza la opción de menú 'Utilidades→Entorno test' (F8) de la biblioteca de funciones o pulsando el botón  en la pantalla inicial.

Ej.: Ejecutar la función 'Z_OBTENER_NUMERO_CLIENTE_XX' desde la biblioteca de funciones.

Para ejecutar un módulo de función desde un programa se utiliza la sentencia CALL FUNCTION <función>. Para que el sistema nos proponga la sentencia de la llamada a una función con todos sus parámetros desde el editor ABAP/4 utilizaremos el botón , marcando la opción 'CALL FUNCTION' y especificando el nombre de la función.

Nota: Los campos que se utilizan en la llamada a un módulo de función deben ser del mismo tipo que los parámetros a los que hacen referencia (definidos en el módulo de función), sino se pueden producir errores en la ejecución del programa.

7. Llamadas a programas y utilización de la memoria.

7.1 Introducción

Para intercambiar datos entre diferentes programas se puede utilizar la memoria SAP y la memoria ABAP/4.

- Memoria SAP.
Es un área de memoria específica para cada usuario que se utiliza para almacenar valores que son retenidos durante toda la sesión del usuario.
- Memoria ABAP/4.
Los valores almacenados en la memoria ABAP/4 solamente son retenidos durante la ejecución de un programa. Esta memoria es utilizada para la transferencia de datos entre dos programas cuando uno de ellos realiza una llamada al otro.

7.2 Parámetros de memoria SAP

En la memoria SAP se pueden almacenar valores asociados a un identificador de tres caracteres, que se mantienen disponibles hasta que finalice la sesión.

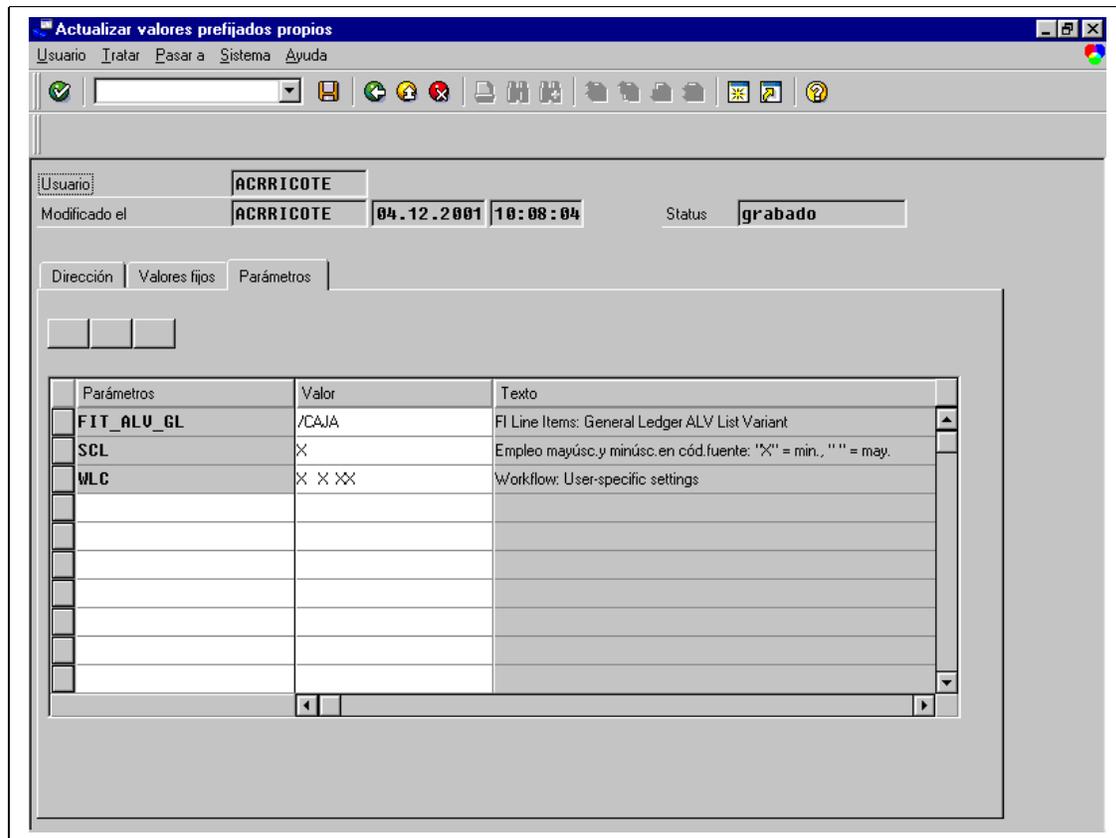
En los campos de las pantallas existen tres atributos relacionados con los parámetros de memoria:

- Id-parám.: Identificador de parámetro para los valores del campo en la memoria SAP. Si el campo está referenciado al diccionario de datos, se informará con el identificador asociado al dominio del campo del diccionario.
- SET Parám.: Al activar este atributo, el sistema almacenará en la memoria SAP el valor que contiene al campo bajo el identificador asociado.
- GET Parám.: Al activar este atributo, el campo se inicializará con el valor definido en la memoria SAP bajo el identificador asociado en lugar de utilizar el valor inicial en función del tipo de dato del campo.

Desde un programa se pueden almacenar y recuperar datos de la memoria SAP con las siguientes sentencias:

- SET PARAMETER.
La sentencia SET PARAMETER ID <identificador> FIELD <campo> almacena el valor del campo en la memoria SAP asociado al identificador especificado.
- GET PARAMETER.
La sentencia GET PARAMETER ID <identificador> FIELD <campo> almacena el valor asociado al identificador en la memoria SAP en el campo especificado.

Los parámetros de memoria se inicializan al comenzar una sesión con los valores almacenados en los parámetros de usuario, que se mantienen desde la opción de menú 'Sistema→Valores prefijados→Datos propios' (SU3).



- o Idp: Identificador de parámetro.
- o Valor parámetro: Valor asociado al identificador.

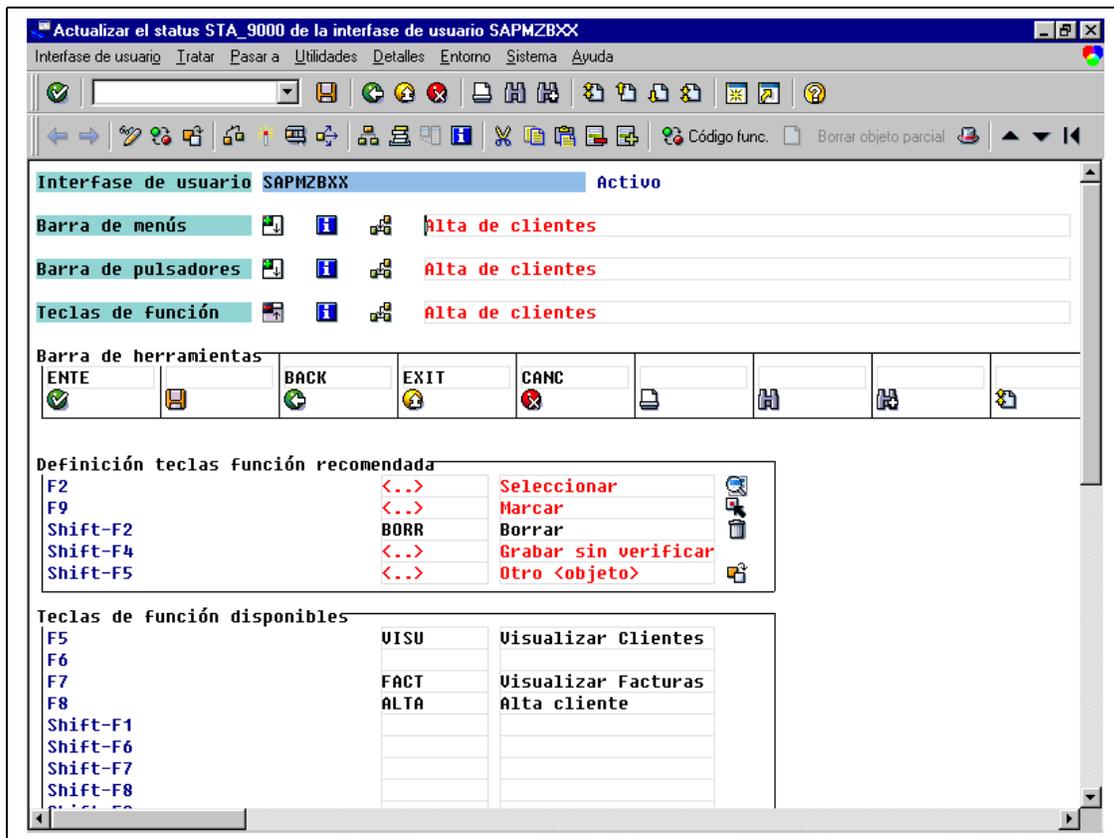
7.3 Sentencias de llamada a programas

Para realizar llamadas a otros programas desde un programa se utilizan las siguientes sentencias:

- o SUBMIT <listado> AND RETURN: Realiza una llamada a un listado. Si no se especifica la cláusula 'AND RETURN' finalizará el programa actual y se ejecutará el listado sin regresar al programa actual.
- o CALL TRANSACTION <transacción>: Realiza una llamada a una transacción.

Ej.: Se incluirá un pulsador en la pantalla de altas de clientes para ejecutar el listado de visualización de facturas de clientes.

Se incluye un pulsador en la barra de pulsadores el Status 'STA_9000' con el código de función 'FACT' de tipo 'E' (Comando Exit) y el texto 'Visualizar clientes' asociado a la tecla de función 'F6'.



Se añade el procesamiento del código de función 'FACT' en el módulo PAI 'SALIR_9000', realizando una llamada al listado 'ZREPO1XX' en caso de que se seleccione la función 'Visualizar facturas'

```
WHEN 'FACT'.                " Visualizar facturas
*   Se ejecuta el listado de facturas de clientes
    SUBMIT ZREPO1XX AND RETURN.
```

Se ejecuta el listado de facturas de clientes desde la pantalla de altas de clientes pulsando el botón .

7.4 Intercambio de datos a través de la memoria ABAP/4

Al realizar una llamada a un programa desde otro programa, se pueden intercambiar datos a través de la memoria ABAP/4 utilizando las siguientes sentencias:

- EXPORT <objeto> TO MEMORY ID <identificador>. Almacena el objeto en la memoria ABAP/4 asociado al identificador especificado, que puede tener una longitud de 32 caracteres. Cada vez que se exportan datos bajo un mismo identificador se sobrescriben los anteriores.
- IMPORT <objeto> FROM MEMORY ID <identificador>. Recupera el objeto asociado al identificador especificado de la memoria ABAP/4.
- FREE MEMORY ID <identificador>. Libera de la memoria ABAP/4 los datos almacenados bajo el identificador especificado. Si no se especifica la cláusula 'ID' se liberará toda la memoria ABAP/4.

8. Interfaces

8.1 Introducción.

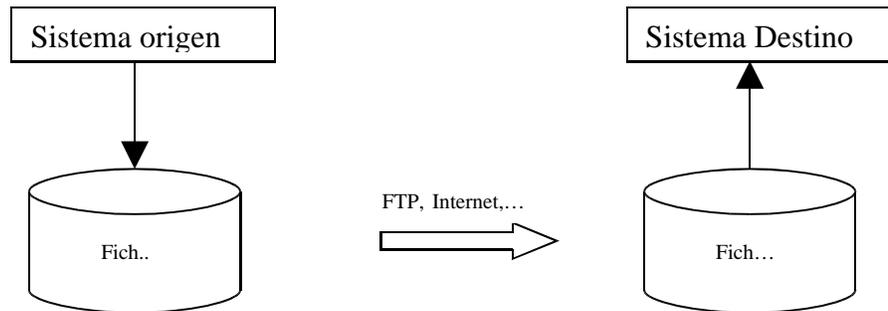
Podemos definir el concepto de interfase como el nexo de unión entre dos sistemas diferentes. Por tanto queda claro que este concepto no es exclusivo de los sistemas de información sino que abarca un espectro más grande.

Si focalizamos nuestra atención en los sistemas de información, podemos darnos cuenta rápidamente, que dentro de una empresa existen diferentes entornos que necesitan comunicarse, necesitan interactuar.

Un claro ejemplo de necesidad de comunicación es la existente entre el entorno SAP y el resto de aplicaciones existentes dentro de una empresa.

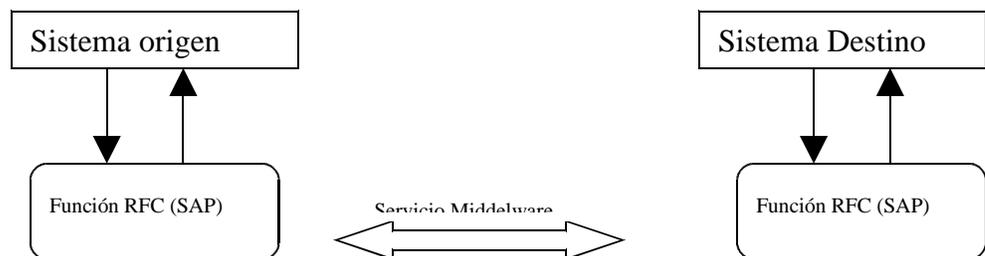
Podríamos hacer varias clasificaciones de interfases:

- En cuanto a su naturaleza :
 - Basadas en ficheros.



El intercambio de la información se realiza a través de ficheros. Así el sistema origen elabora el fichero , con un determinado formato, y se lo envía al sistema destino que lo procesa.

- Basadas en funciones RFC (Remote Function Call).
El intercambio de información se realiza a través de aplicaciones que unen virtualmente los sistemas y ponen a disposición los datos del sistema origen en el sistema destino.



- En cuanto a su modo de ejecución:
 - On-line.
Podemos decir que una interfase es on-line cuando el proceso que la controla es on-line . Normalmente las funciones de naturaleza RFC , serán on-line.
 - Batch.
Una interfase es Batch cuando , el proceso que la controla es BATCH. Las interfases basadas en ficheros irán asociadas a procesos batch.

8.2 Tratamiento de ficheros.

8.2.1 Instrucciones básicas.

Para poder trabajar con ficheros que se encuentren en el servidor de aplicación, tenemos las siguientes instrucciones básicas:

- **OPEN DATASET nfich .**
Abre un fichero para su tratamiento, si no se añade ninguna opción abre el fichero para lectura en modo binario.
Las principales opciones son:
 - **.. FOR OUTPUT** Abre el fichero para escritura. Si el fichero existe lo borra y lo crea nuevamente.
 - **.. FOR INPUT** Abre el fichero para lectura.
 - **.. FOR APPENDING.**
Abre un fichero para escritura si el fichero no existe se crea. Si ya existe, se comenzará a escribir al final del fichero.
 - **..IN BINARY MODE.**
Abre el fichero en modo binario , es decir, la información se tratará por bloques. De tal modo que en cada lectura se leerán /escribirán n caracteres.
 - **..IN TEXT MODE.**
Abre el fichero en modo texto, es decir, la información se tratará línea a línea. Cada lectura/escritura procesará una línea del fichero.
 - **..MESSAGE d_mes**
Almacena en 'd_mes' el mensaje devuelto por el sistema operativo.
 - **..AT POSITION d_pos.**
Sitúa el puntero de escritura en la posición d_pos (bytes) desde el inicio del fichero. (Esta opción , no tiene sentido emplearla para tratamientos en modo texto).
 - **.. TYPE** Permite definir atributos del fichero.
 - **.. FILTER** Permite ejecutar comandos propios del sistema operativo.
- **READ DATASET nfich INTO wa.**

Lee un bloque de información del fichero:

Si el fichero se abre en modo TEXTO:

Se lee una línea completa de tal forma que si el campo 'wa' es de menor tamaño que la línea , se perderá su contenido , en caso contrario se completará con blancos. El puntero de lectura se situará al comienzo de la línea siguiente.

Si el fichero se abre en modo BINARIO:

Se lee del fichero tantos bytes como tamaño tenga el campo 'wa'.

- TRANSFER wa TO nfich.

Escribe el contenido del campo WA en el fichero abierto para escritura.

Si en modo texto:

Escribe una línea completa.

Si en modo binario:

Escribe tantos bytes como tenga el campo 'wa'.

- CLOSE DATASET nfich.

Cierra un fichero abierto para lectura o escritura.

- DELETE DATASET nfich.

Borra el fichero nfich.

8.2.2 Creación de ficheros.

Vamos a ver un ejemplo de como leer y escribir registros en un fichero.

El ejemplo va a consistir en volcar a un fichero, nuestra tabla maestro de clientes con todos sus campos.

La estructura del fichero será así:

Campo	Tipo	Long.	Descripción
BUKRS	CHAR	4	Sociedad
NCLIE	CHAR	10	Número de cliente
NOMBR	CHAR	20	Nombre cliente
APEL1	CHAR	25	Primer apellido
APEL2	CHAR	25	Segundo apellido
FNACI	DATS	8	Fecha de nacimiento

Crearemos un programa tipo 'REPORT' que llamaremos 'ZINTERXX'.

Como parámetros de selección pondremos la sociedad obligatoria y como rango de selección el nº de cliente.

Añadiremos un nuevo parámetro obligatorio que representará el nombre del fichero.

El fichero que vamos a crear va a ser en modo texto.

El código del programa será algo así:

```
*****
* PROGRAMA: ZINTERXX *
* DESCRIPCION: Vuelca los registros seleccionados de la tabla *
* maestro de clientes ZCLIENXX a un fichero de texto. *
* AUTOR : MAD00 FECHA: 28/08/2001 *
*-----*
* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *
*****
REPORT ZINTERXX.
*****
* Tablas del diccionario de datos *
*****
TABLES: ZCLIENXX. " Maestro de clientes.

*****
* Tablas internas *
*****
* Tabla interna de selección de clientes
DATA: BEGIN OF I_ZCLIENXX OCCURS 0,
      BUKRS LIKE ZCLIENXX-BUKRS, "Sociedad
      NCLIE LIKE ZCLIENXX-NCLIE, "Número de cliente
      NOMBR LIKE ZCLIENXX-NOMBR, "Nombre cliente
      APEL1 LIKE ZCLIENXX-APEL1, "Primer apellido
      APEL2 LIKE ZCLIENXX-APEL2, "Segundo apellido
      FNACI LIKE ZCLIENXX-FNACI, "Fecha de nacimiento
      END OF I_ZCLIENXX.
*****
* Definición de variables globales *
*****
DATA :D_MERROR(100) TYPE C. " Mensaje de error

*****
* Pantalla de selección *
*****
* Parámetros
SELECTION-SCREEN BEGIN OF BLOCK BLOQ1 WITH FRAME TITLE TEXT-001.
PARAMETERS P_BUKRS LIKE ZCLIENXX-BUKRS OBLIGATORY
           DEFAULT '0001'. " Sociedad
SELECT-OPTIONS:
      S_NCLIE FOR ZCLIENXX-NCLIE. " Nº de cliente
SELECTION-SCREEN END OF BLOCK BLOQ1.
* Parámetro nombre de fichero destino.
SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE TEXT-002.
* Nombre del fichero (activamos minúsculas permitidas).
PARAMETERS P_NFICH(50) TYPE C OBLIGATORY DEFAULT
'usr/sap/tmp/zcliexx.dat' LOWER CASE
.
```

SELECTION-SCREEN END OF BLOCK BLOQ2.

```

*****
* Comienzo de selección *
*****
START-OF-SELECTION.
* Seleccionamos los datos de los clientes que cumplen criterios
  SELECT *
  FROM ZCLIENXX INTO CORRESPONDING FIELDS OF TABLE I_ZCLIENXX
  WHERE BUKRS = P_BUKRS
  AND  NCLIE IN S_NCLIE.
* Comprobamos si hay datos seleccionados en cuyo caso ,
* crearemos el fichero
  IF ( SY-SUBRC = 0 ).
* Abrimos el fichero para escritura en modo texto
OPEN DATASET P_NFICH FOR OUTPUT IN TEXT MODE MESSAGE D_MERROR.
* Comprobamos si no hubo error
  IF ( SY-SUBRC = 0 ).
* Para cada uno de los registros seleccionados
  LOOP AT I_ZCLIENXX.
* Escribimos en el fichero
  TRANSFER I_ZCLIENXX TO P_NFICH.
ENDLOOP.
* Cerramos el fichero
  CLOSE DATASET P_NFICH.
* Mostramos mensaje 'Fichero creado con éxito'.
  MESSAGE I000(38) WITH TEXT-003.
* Si hay error , mostramos el mensaje, con un mensaje de error
  ELSE.
  MESSAGE E000(38) WITH D_MERROR.
  ENDIF.
* Si no hay datos seleccionados.
  ELSE.
* Mostramos mensaje 'No se seleccionaron datos'.
  MESSAGE E000(38) WITH TEXT-004.
  ENDIF.

```

8.2.3 Lectura de Ficheros.

Vamos a continuación a realizar un programa (ZINTER1XX) que muestre un listado con el contenido de un fichero. Dando la posibilidad de borrar el fichero una vez procesado.

```

*****
* PROGRAMA: ZINTE1XX *
* DESCRIPCION: Muestra un listado con el contenido de un fichero *
* AUTOR : MAD00          FECHA: 28/08/2001 *
*-----*

```

```

* CONTROL DE MODIFICACIONES *
* FECHA.      AUTOR.      DESCRIPCION MODIFICACION. *
*****
REPORT ZINTE1XX NO STANDARD PAGE HEADING.

*****
* Definición de variables globales *
*****
DATA :D_MERROR(100) TYPE C,      " Mensaje de error
D_LINEA(255) TYPE C.      " Línea de fichero.

*****
* Pantalla de selección *
*****

* Parámetro nombre de fichero
SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE
TEXT-002.
* Nombre del fichero (activamos minúsculas permitidas).
PARAMETERS: P_NFICH(50) TYPE C OBLIGATORY DEFAULT
'usr/sap/tmp/zclienxx.dat' LOWER CASE,
            P_DELETE AS CHECKBOX. " Flag de borrado fichero
SELECTION-SCREEN END OF BLOCK BLOQ2.

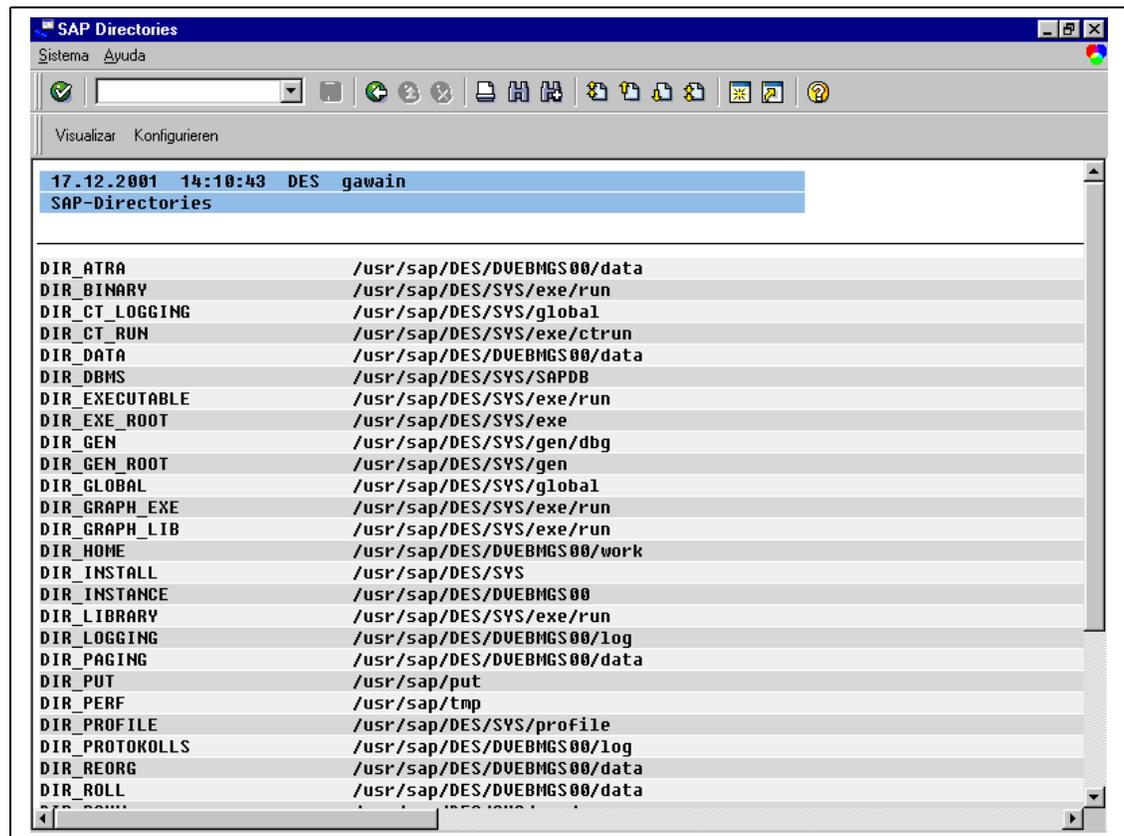
*****
* Comienzo de selección *
*****
START-OF-SELECTION.
* Abrimos el fichero para lectura en modo texto.
OPEN DATASET P_NFICH FOR INPUT IN TEXT MODE MESSAGE
D_MERROR.
* Comprobamos si no hubo error
IF ( SY-SUBRC = 0 ).
* Leemos el primer registro del fichero
READ DATASET P_NFICH INTO D_LINEA.
* Mientras no haya error de lectura
WHILE ( SY-SUBRC = 0 ).
* Escribimos la línea leída
WRITE / D_LINEA.
* Leemos el siguiente registro
READ DATASET P_NFICH INTO D_LINEA.
ENDWHILE.
* Una vez procesado el fichero, borramos el fichero si está
* marcado el flag de borrado
CHECK NOT ( P_DELETE IS INITIAL ).
DELETE DATASET P_NFICH.
* Comprobamos si se pudo borrar
IF ( SY-SUBRC = 0 ).

```

- * *Mostramos mensaje de 'Fichero borrado'.*
MESSAGE S000(38) WITH TEXT-005.
ENDIF.
- * *Si hay error , mostramos el mensaje, con un mensaje de error*
ELSE.
MESSAGE E000(38) WITH D_MERROR.
ENDIF.

8.2.4 Explorar ficheros.

Para poder visualizar los ficheros existentes en el servidor de aplicación disponemos de la transacción estandar AL11 que nos permite explorar el contenido del directorio /usr/sap.



Haciendo doble-click sobre el directorio a explorar *'/usr/sap/tmp'* en nuestro caso podremos visualizar los ficheros existentes así como el contenido de los mismos.

Nota: Esta es una herramienta bastante limitada ya que únicamente nos dejará visualizar el contenido de esta ruta (/usr/sap) , normalmente dispondremos de otras herramientas para poder visualizar y tratar ficheros en el servidor de aplicación.

8.3 Batch Input .

8.3.1 Introducción.

En capítulos anteriores , hemos visto como se introducen datos, que actualizan las bases de datos, a través de pantallas o dynpros que se llaman dentro de una transacción . Introducimos los datos , navegando por cada una de las pantallas completando los datos en los campos y pulsando las opciones oportunas (pantalla siguiente , grabar ...). En todo este proceso , es necesario la intervención de un usuario.

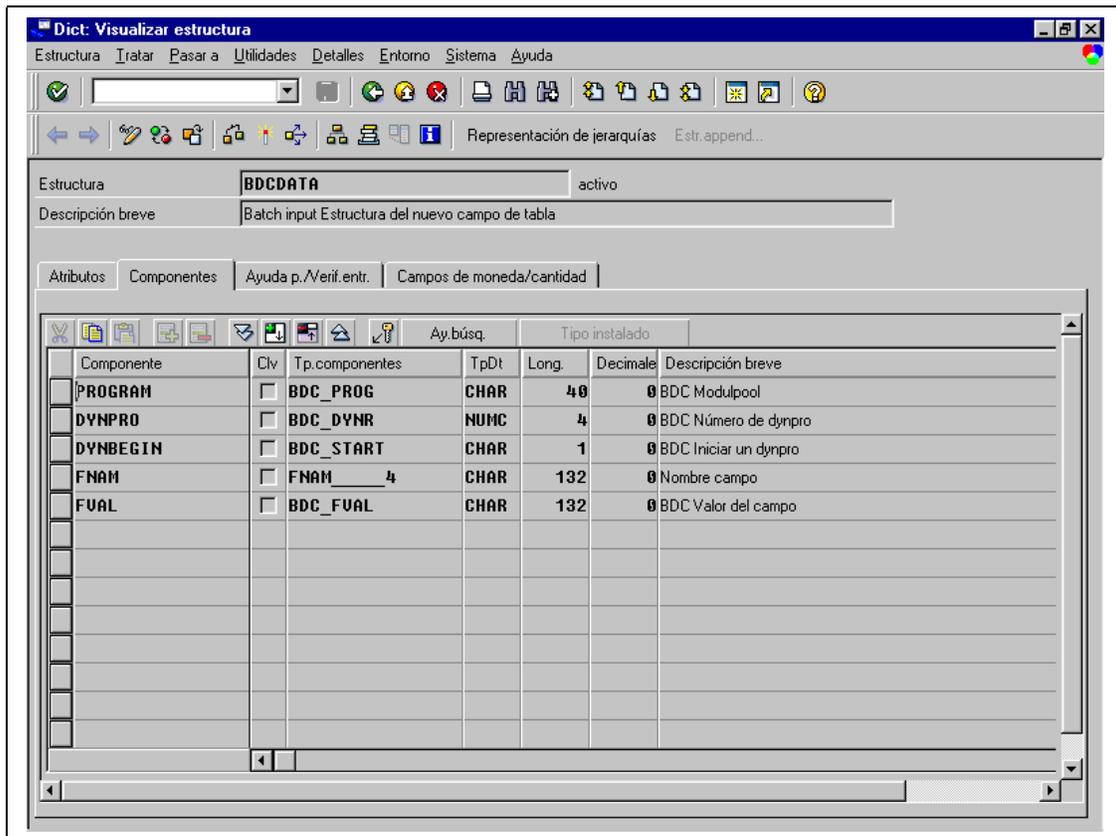
El sistema Batch Input o BDC (Batch Data Communication) para introducir datos en la base de datos de SAP tiene las siguientes características:

- Puede procesar grandes volúmenes de datos.
- Se puede planificar y ejecutar como proceso de fondo o como proceso on-line.
- Se puede procesar sin la intervención de un usuario.
- Permite usar transacciones como si los datos se introdujeran de forma manual, pero realizando idénticas comprobaciones de integridad, autorizaciones...

En este sistema de introducción de datos , es necesario tener un fichero especial que llamaremos 'Juego de datos' en el que se tiene perfectamente estructurada la siguiente información:

- Por una parte la información específica a introducir .
- Por otra parte los comandos , pantallas, nombres de campo ... en las que introducir la información.

Estos juegos de datos tienen siempre la misma forma definida en la estructura estándar 'BDCDATA' :



Construir el 'Juego de datos' consiste en ir añadiendo registros secuencialmente , en los que se definen las secuencias de pantalla necesarias y datos para lograr su almacenamiento en las tablas SAP.

Tendremos dos tipos de registros (aunque con la misma estructura).

- Registros de Identificación de programa y dynpro.
En este tipo de registros , se detalla el nombre del programa y nº de dynpro que va a contener los campos y los valores que van a tomar los registros siguientes.
- Registros de campos y valores .
Este tipo de registros , contienen el nombre de los campos del dynpro así como los datos específicos que se deben 'introducir' en ellos. Un registro especial dentro de este tipo es el que contiene el valor del comando de dynpro. BDC_OKCODE.

La secuencia de registros BDC será algo así:

Curso programación ABAP IV	
-----------------------------------	--

PROGRAM	DYNPRO	DYNBEGIN	FNAM	FVAL
Programa1	1ddd	X		
			Campo1d	Valor campo1d
			Campo2d	Valor campo 2d
			BDC_OKCODE	Comando
Programa2	2ccc	X		
			Campo 1c	Valor campo 1c
			Campo 2c	Valor campo 2c
			BDC_OKCODE	Comando
....				
Programa n	Njjj	X		
			Campo 1c	Valor campo 1c
			Campo 2c	Valor campo 2c
			BDC_OKCODE	Comando

Para almacenar estos registros en un fichero especial al que denominamos 'Juego de datos' disponemos de tres funciones estándares que nos facilitan esta tarea:

- **BDC_OPEN_GROUP.**

Nos permite abrir la sesión para la creación del juego de datos y que éste se almacene en la cola de juegos de datos pendientes.

Los parámetros principales son:

GROUP: Nombre con el que se almacenará el juego de datos.

USER: Usuario con el que se ejecutará el juego de datos.

HOLDDATE: Fecha a partir de la cual el juego de datos puede ser ejecutado.

KEEP: Flag de indicador de borrado del juego de datos tras su ejecución con éxito. (' permanece , 'X' no se borra).

- **BDC_INSERT.**

Permite incluir los registros al juego de datos a través de una tabla interna.

Hay que indicar los parámetros.

TCODE: Código de transacción a ejecutar.

DYNPROTAB: Tabla interna que contiene los registros.

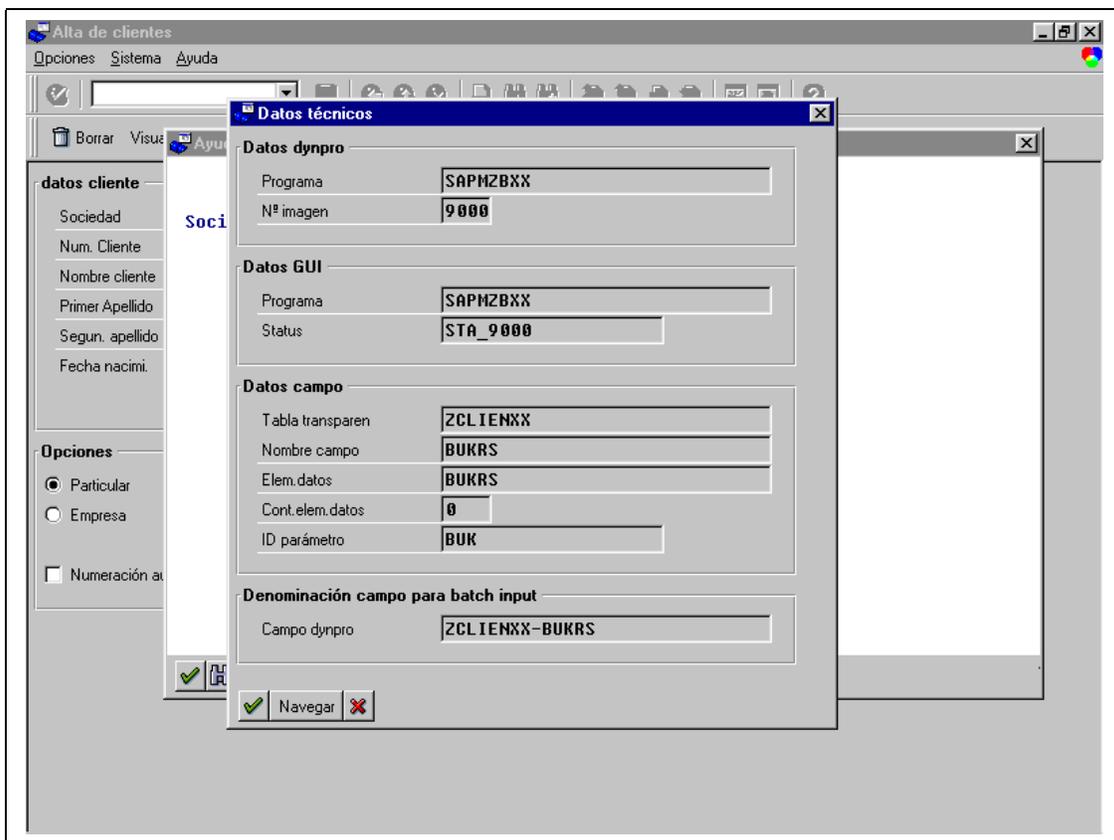
- **CLOSE_GROUP.**

Cierra el Juego de datos.

Una vez creado el fichero BDC o 'Juego de datos', este quedará almacenado para su posterior tratamiento: Ejecución , borrado

Para saber el nombre del programa ,dynpro , nombre de los campos con los que debemos completar la tabla DBC tenemos dos posibilidades:

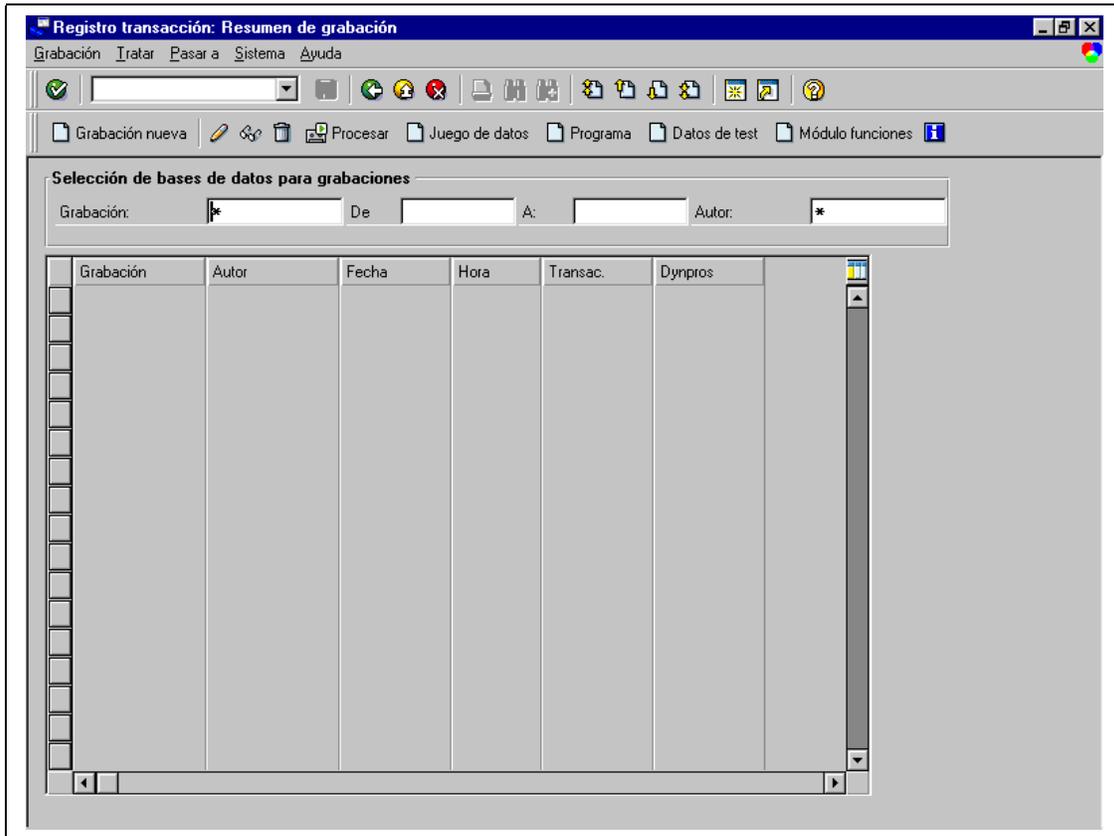
- Obtener el nombre de los campos mediante la ayuda on-line disponible en todas las pantallas.
Para obtener los datos de un campo , situados sobre el campo de pantalla pulsaremos 'F1', nos saldrá una ventana mostrando la descripción del campo , esta ventana tiene una opción pulsándolo nos aparecerá una ventana con este aspecto:



Donde se muestra información sobre el programa , nombre del campo ...

- Utilizar la utilidad de 'grabar' una transacción.
Podemos acceder a ella a través del menú Sistema (Existirá en todos los menús) seguiremos la ruta:
Servicios→Batch Input → Grabadora (SHDB).

Llegaremos a la siguiente pantalla :



En esta pantalla podremos crear grabaciones , visualizar existentes,

Para crear una nueva , pulsaremos el botón  deberemos indicar un nombre para la grabación 'ALTAS' e indicaremos el código de la transacción 'ZBXX' . A continuación pulsaremos el botón  nos saldrá la transacción preparada por introducir los datos, introduciremos los datos correspondientes (del mismo modo que si estuviéramos en la transacción directamente) al finalizar nos saldrá una pantalla con el siguiente aspecto:

Línea	Programa	Dynpro	Indic	Nom.cpo.	valor de campo
1			T	ZBXX	
2	SAPHZBXX	9000	X		
3				BDC_CURSOR	ZCLIENXX-NCLIE
4				BDC_ORCODE	=ENTE
5				ZCLIENXX-BURRS	0001
6				ZCLIENXX-NCLIE	0000000022
7				ZCLIENXX-NOMBR	ADOLFO
8				ZCLIENXX-APEL1	LOPEZ
9				ZCLIENXX-APEL2	SANCHEZ
10				ZCLIENXX-FNACI	20010101
11				D_PARTICULAR	X
12	SAPHZBXX	9000	X		
13				BDC_CURSOR	ZCLIENXX-NOMBR
14				BDC_ORCODE	=ALTA
15				ZCLIENXX-NOMBR	ADOLFO
16				ZCLIENXX-APEL1	LOPEZ
17				ZCLIENXX-APEL2	SANCHEZ
18				ZCLIENXX-FNACI	20.01.0101
19				D_PARTICULAR	X

Línea 1 - 19 de 22

Donde podremos visualizar el nombre del programa , campos , comandos ...

8.3.2 Creación del juego de datos.

Suponemos el siguiente caso. Tenemos que dar de alta a todos los clientes en nuestra tabla maestro de clientes ZCLIENXX , para ello , disponemos:

- De la transacción 'ZBXX'(alta de clientes que creamos en capítulos anteriores).
- Tenemos la información de los clientes almacenada en un fichero '/usr/sap/tmp/zlienxx.dat'. (Cuya estructura es la que vimos en el apartado anterior).

Vamos a crear un programa 'ZINTE2XX' que lea los datos del fichero y elabore un juego de datos para dar de alta clientes .

El nombre del juego de datos y el nombre del fichero , serán parámetros del programa.

El código del programa será algo así:

```
*****
* PROGRAMA: ZINTE2XX *
```

```

* DESCRIPCION: Genera un Juego de datos de Alta de clientes , con *
* los datos de los clientes que lee de un fichero. *
* AUTOR : MAD00 FECHA: 28/08/2001 *
*-----*
* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *
*****
REPORT ZINTE2XX NO STANDARD PAGE HEADING.
*****
* Tablas del diccionario de datos *
*****
TABLES: ZCLIENXX. " Maestro de clientes.
*****
* Definición de tablas internas *
*****
* Tabla interna para almacenar los registros del fichero
DATA: BEGIN OF I_ZCLIENXX OCCURS 0,
      BUKRS LIKE ZCLIENXX-BUKRS, "Sociedad
      NCLIE LIKE ZCLIENXX-NCLIE, "Número de cliente
      NOMBR LIKE ZCLIENXX-NOMBR, "Nombre cliente
      APEL1 LIKE ZCLIENXX-APEL1, "Primer apellido
      APEL2 LIKE ZCLIENXX-APEL2, "Segundo apellido
      FNACI LIKE ZCLIENXX-FNACI, "Fecha de nacimiento
      END OF I_ZCLIENXX.
* Tabla interna para almacenar los registros del juego de datos
DATA: BEGIN OF I_BDCTAB OCCURS 0.
      INCLUDE STRUCTURE BDCDATA.
      DATA: END OF I_BDCTAB.
*****
* Definición de variables globales *
*****
DATA :D_MERROR(100) TYPE C. " Mensaje de error
*****
* Pantalla de selección *
*****

* Parámetro nombre de fichero
SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE TEXT-002.
* Parámetros
PARAMETERS:
* Nombre del fichero (activamos minúsculas permitidas).
      P_NFICH(50) TYPE C OBLIGATORY DEFAULT '/usr/sap/tmp/zclienxx.dat'
      LOWER CASE,
* Nombre del juego datos
      P_JDATOS(12) TYPE C OBLIGATORY DEFAULT 'ZCLIENXX'.

SELECTION-SCREEN END OF BLOCK BLOQ2.
*****

```

```

* Comienzo de selección
*****
START-OF-SELECTION.
* Abrimos el fichero para lectura en modo texto.
OPEN DATASET P_NFICH FOR INPUT IN TEXT MODE MESSAGE D_MERROR.
* Comprobamos si no hubo error
IF ( SY-SUBRC = 0 ).
* Inicializamos la tabla interna
REFRESH I_ZCLIENXX.
CLEAR I_ZCLIENXX.
* Leemos el primer registro del fichero sobre la tabla interna
READ DATASET P_NFICH INTO I_ZCLIENXX.
* Mientras no haya error de lectura
WHILE ( SY-SUBRC = 0 ).
* Añadimos el registro leído a la tabla interna
APPEND I_ZCLIENXX.
* Inicializamos la cabecera
CLEAR I_ZCLIENXX.
* Leemos el siguiente registro
READ DATASET P_NFICH INTO I_ZCLIENXX.
ENDWHILE.
* Creamos el juego de datos con el contenido de la tabla I_ZCLIENXX
PERFORM CREAR_JUEGO_DATOS.
* Mostramos mensaje de 'Juego de datos creado'.
MESSAGE I000(38) WITH TEXT-003 P_JDATOS.
* Si hay error , mostramos el mensaje, con un mensaje de error
ELSE.
MESSAGE E000(38) WITH D_MERROR.
ENDIF.

```

```

*****
Rutinas adicionales.
*****
*&-----*
*& Form CREAR_JUEGO_DATOS
*&-----*
* Crea un juego de datos con el nombre indicado por el parámetro *
* P_JDATOS, para la transacción ZBXX con el contenido de la *
* tabla interna I_ZCLIENXX.
*-----*
FORM CREAR_JUEGO_DATOS.

```

```

*****
*Definición de variables locales
*****
DATA: L_FNACI(10) TYPE C. " Formateo de fecha para pantalla.
*****
* Proceso

```

* Comprobamos que la tabla interna I_ZCLIENXX tiene registros
CHECK NOT (I_ZCLIENXX[] IS INITIAL).

* Abrimos el juego de datos.

CALL FUNCTION 'BDC_OPEN_GROUP'

EXPORTING

* CLIENT = SY-MANDT
* DEST = FILLER8
GROUP = P_JDATOS
* HOLDDATE = FILLER8
KEEP = ''
USER = SY-UNAME
* RECORD = FILLER1
* IMPORTING
* QID =

EXCEPTIONS

CLIENT_INVALID = 1
DESTINATION_INVALID = 2
GROUP_INVALID = 3
GROUP_IS_LOCKED = 4
HOLDDATE_INVALID = 5
INTERNAL_ERROR = 6
QUEUE_ERROR = 7
RUNNING = 8
SYSTEM_LOCK_ERROR = 9
USER_INVALID = 10
OTHERS = 11.

* Continuamos si no hay error

CHECK (SY-SUBRC = 0).

* Completamos la tabla BDC de la forma oportuna

LOOP AT I_ZCLIENXX.

* Formateamos el campo fecha para salida (dd.mm.aaaa)

* en la pantalla no podemos indicar el formato interno (aaaammdd)

WRITE I_ZCLIENXX-FNACI TO L_FNACI.

* Completamos la pantalla inicial con los datos del cliente actual

PERFORM BDC_INSERT USING:

'X' 'SAPMZBXX' '9000',
'' 'ZCLIENXX-BUKRS' I_ZCLIENXX-BUKRS,
'' 'ZCLIENXX-NCLIE' I_ZCLIENXX-NCLIE,
'' 'ZCLIENXX-NOMBR' I_ZCLIENXX-NOMBR,
'' 'ZCLIENXX-APEL1' I_ZCLIENXX-APEL1,
'' 'ZCLIENXX-APEL2' I_ZCLIENXX-APEL2,
'' 'ZCLIENXX-FNACI' L_FNACI,
'' 'D_PARTICULAR' 'X',
'' 'BDC_OKCODE' 'ENTE',

* En la pantalla inicial (con datos introducidos) damos alta

'X' 'SAPMZBXX' '9000',

```

'' 'BDC_OKCODE'   'ALTA'.
ENDLOOP.
* Introducimos el código de fin de transacción después del
* último cliente
PERFORM BDC_INSERT USING:
* En la pantalla inicial (abandonamos la transacción).
  'X' 'SAPMZBXX'   '9000',
  '' 'BDC_OKCODE'  '/ECANC'.
* Una vez completado la transacción , la incluimos en el J.datos
CALL FUNCTION 'BDC_INSERT'
  EXPORTING
    TCODE          = 'ZBXX'
*   POST_LOCAL     = NOVBLOCAL
*   PRINTING       = NOPRINT
  TABLES
    DYNPROTAB      = I_BDCTAB
  EXCEPTIONS
INTERNAL_ERROR = 1
  NOT_OPEN       = 2
  QUEUE_ERROR    = 3
  TCODE_INVALID = 4
  PRINTING_INVALID = 5
  POSTING_INVALID = 6
  OTHERS         = 7.
* Comprobamos si hubo error, mostramos mensaje de error
IF ( SY-SUBRC <> 0 ).
  MESSAGE E000(38) WITH SY-SUBRC.
ENDIF.
* Una vez completado el Juego de datos , lo cerramos
CALL FUNCTION 'BDC_CLOSE_GROUP'
  EXCEPTIONS
    NOT_OPEN     = 1
    QUEUE_ERROR  = 2
    OTHERS       = 3.
ENDFORM.                " CREAR_JUEGO_DATOS
*&-----*
*&   Form BDC_INSERT
*&-----*
*   Inserta en la tabla interna global I_BDCTAB los campos recibido *
* Distinguiendo si se trata de un registro de dynpro o de un registro *
* de campo.
*-----*
*   -->PE_DYNBEGIN   Flag de pantalla/ campo
*   -->PE_FNAM      Nombre de programa / campo
*   -->PE_FVAL      Valor pantalla / campo
*-----*
FORM BDC_INSERT USING VALUE(PE_DYNBEGIN)
                      VALUE(PE_FNAM)

```

VALUE(PE_FVAL).

* Proceso

*

* Inicializamos la cabecera de la tabla

CLEAR I_BDCTAB.

* Informamos el flag de pantalla

I_BDCTAB-DYNBEGIN = PE_DYNBEGIN.

* En función si es o no pantalla

IF (PE_DYNBEGIN = 'X').

I_BDCTAB-PROGRAM = PE_FNAM.

I_BDCTAB-DYNPRO = PE_FVAL.

* Si se trata de un campo

ELSE.

I_BDCTAB-FNAM = PE_FNAM.

I_BDCTAB-FVAL = PE_FVAL.

ENDIF.

* Insertamos el registro en la tabla interna

APPEND I_BDCTAB.

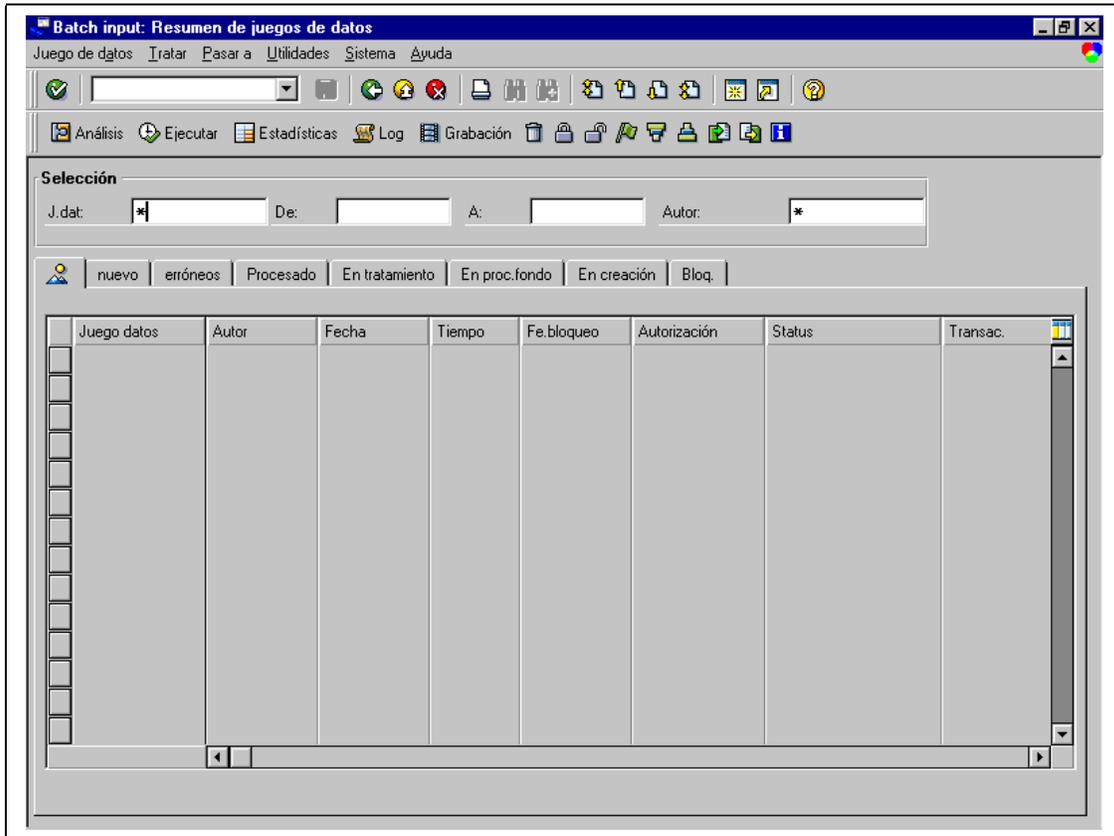
ENDFORM. " BDC_INSERT

1.3.3 Tratamiento de un juego de datos.

Ya hemos generado un juego de datos , que estará almacenado en algún lugar del sistema, vamos a ver ahora como visualizan, procesan, borran ...

En el menú Sistema (Existirá en todos los menús) seguiremos la ruta:
Servicios→Batch Input → Carpetas (SM35).

Llegaremos a la siguiente pantalla:



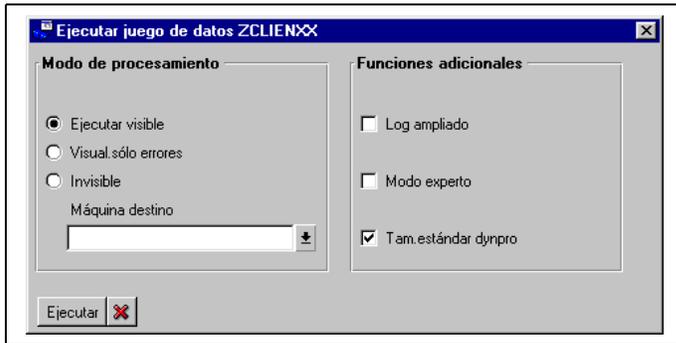
Donde podremos seleccionar los juegos de datos , por nombre y fecha de creación así como por su Status de proceso....

Para ver nuestro juego de datos pondremos el nombre dado (*ZCLIENXX*).

Aparecerá nuestro juego de datos, en esta pantalla , podremos realizar las siguientes funciones:

Dentro del menú 'Juego de datos'.

- Ejecutar Juego de datos (F8).
Nos permite ejecutar el juego de datos ofreciendo varias posibilidades de ejecución a través de la ventana:



Ejecutar visible: Se visualizan las pantallas por las que va pasando proceso deteniéndose en cada una de ellas hasta que se pulse enter.

Visualizar sólo errores Únicamente se visualizará la secuencia de pantallas cuando se encuentre un error .

Ejecutar Invisible: Tendrá el mismo efecto que la opción 'Procesar en Fondo'

- Procesar en Fondo (F5).
Ejecuta el juego de datos en un proceso en fondo (Batch). Para ello genera un JOB con el mismo nombre del juego de datos.(Podremos indicar el nombre de la máquina en la que queremos que se ejecute).
- Borrar (Shift + F2). 
Borra el juego de datos.

Dentro del menú 'Pasar a' (o botones):

- Análisis juego datos. 
Permite visualizar el contenido, pudiendo visualizar el nombre de las pantallas así como su contenido.
- Estadísticas. 
Una vez ejecutado el juego de datos , muestra un informe con las transacciones correctas , erróneas ...
- Log. 
Muestra un informe con el log de la ejecución , se crearán tantos logs como ejecuciones del juego de datos se realicen.

Ejecutaremos el juego de datos. A continuación comprobamos que las entradas se han dado de alta en la tabla.

8.4 Call transaction.

La filosofía del Call Transaction es idéntica a la del juego de datos , se trata de crear un conjunto de registros de pantallas y de datos para una transacción

determinada con la misma estructura que para el juego de datos que se ejecuta de forma inmediata.

Las principales diferencias son:

- No se genera un juego de datos que se almacena en forma de fichero especial.
- La ejecución es inmediata.
- No se pueden incluir datos para dos transacciones distintas.

Por tanto para introducir datos mediante Call Transaction solamente necesitaremos completar la tabla de registros I_BDCTAB del mismo modo que en un Batch Input, una vez completada llamaremos a su ejecución mediante la instrucción:

CALL TRANSACTION tcod.

Tiene las siguientes opciones:

- .. AND SKIP FIRST SCREEN.
Permite saltar la primera pantalla de la transacción , esto será útil para determinadas transacciones que la pantalla inicial sea informativa o no se recojan datos.
- ..USING itab
Permite indicar la tabla interna que contendrá los registros con la estructura BDCDATA.
- ... MODE mode
Permite señalar el modo de llamada los valores posibles son:
 - 'A' Visualizando las pantallas.
 - 'E' Visualizar pantallas sólo si hay error.
 - 'N' No visualizar pantallas.
- ... UPDATE upd
Permite señalar la manera en que se actualizarán los datos en las tablas los valores posibles son:
 - 'A' Actualización asíncrona.
 - 'S' Actualización síncrona.
- ... MESSAGES INTO messtab
Permite recoger en la tabla interna 'messtab' los mensajes que se produzcan durante la ejecución de la transacción.

Vamos a modificar nuestro programa anterior 'ZINTE2XX' para que de la posibilidad de introducir los datos leídos del fichero bien generando un juego de datos o bien ejecutando Call transaction.

Para ello realizaremos las siguientes modificaciones:

En primer lugar , definiremos dos casillas de selección que den la posibilidad de seleccionar un método u otro. (Si se selecciona juego de datos , el parámetro que indica el nombre del juego de datos a crear será obligatorio).

El código resultante será algo así:

```
*****
* PROGRAMA: ZINTE2XX *
* DESCRIPCION: Con los datos de los clientes leídos de un fichero *
* da la posibilidad de crear un juego de datos o cargar *
* directamente los datos con Call transaction. *
* AUTOR : MAD00 FECHA: 28/08/2001 *
*-----*
* CONTROL DE MODIFICACIONES *
* FECHA. AUTOR. DESCRIPCION MODIFICACION. *
*****
REPORT ZINTE2XX NO STANDARD PAGE HEADING.
*****
* Tablas del diccionario de datos *
*****
TABLES: ZCLIENXX. " Maestro de clientes.
*****
* Definición de tablas internas *
*****
* Tabla interna para almacenar los registros del fichero
DATA: BEGIN OF I_ZCLIENXX OCCURS 0,
      BUKRS LIKE ZCLIENXX-BUKRS, "Sociedad
      NCLIE LIKE ZCLIENXX-NCLIE, "Número de cliente
      NOMBR LIKE ZCLIENXX-NOMBR, "Nombre cliente
      APEL1 LIKE ZCLIENXX-APEL1, "Primer apellido
      APEL2 LIKE ZCLIENXX-APEL2, "Segundo apellido
      FNACI LIKE ZCLIENXX-FNACI, "Fecha de nacimiento
      END OF I_ZCLIENXX.
* Tabla interna para almacenar los registros del juego de datos
DATA: BEGIN OF I_BDCTAB OCCURS 0.
      INCLUDE STRUCTURE BDCDATA.
DATA: END OF I_BDCTAB.
*****
* Definición de variables globales *
*****
DATA :D_MERROR(100) TYPE C. " Mensaje de error
*****
* Pantalla de selección *
*****
* Parámetro nombre de fichero
```

SELECTION-SCREEN BEGIN OF BLOCK BLOQ1 WITH FRAME TITLE
TEXT-001.

* *Parámetros*

PARAMETERS:

* *Nombre del fichero (activamos minúsculas permitidas).*

P_NFICH(50) TYPE C OBLIGATORY DEFAULT '/usr/sap/tmp/zclienxx.dat'
LOWER CASE.

SELECTION-SCREEN END OF BLOCK BLOQ1.

SELECTION-SCREEN BEGIN OF BLOCK BLOQ2 WITH FRAME TITLE
TEXT-002.

PARAMETERS:

* *Flag de crear juego de datos*

P_JDFL RADIOBUTTON GROUP 001,

* *Nombre del juego de datos.*

P_JDATOS(12) TYPE C DEFAULT 'ZCLIENXX'.

SELECTION-SCREEN SKIP 1.

PARAMETERS

* *Flag de call transaction.*

P_CTFL RADIOBUTTON GROUP 001.

SELECTION-SCREEN END OF BLOCK BLOQ2.

* *Comienzo de selección*

*

START-OF-SELECTION.

* *Abrimos el fichero para lectura en modo texto.*

OPEN DATASET P_NFICH FOR INPUT IN TEXT MODE MESSAGE
D_MERROR.

* *Comprobamos si no hubo error*

IF (SY-SUBRC = 0).

* *Inicializamos la tabla interna*

REFRESH I_ZCLIENXX.

CLEAR I_ZCLIENXX.

* *Leemos el primer registro del fichero sobre la tabla interna*

READ DATASET P_NFICH INTO I_ZCLIENXX.

* *Mientras no haya error de lectura*

WHILE (SY-SUBRC = 0).

* *Añadimos el registro leído a la tabla interna*

APPEND I_ZCLIENXX.

* *Inicializamos la cabecera*

CLEAR I_ZCLIENXX.

* *Leemos el siguiente registro*

READ DATASET P_NFICH INTO I_ZCLIENXX.

ENDWHILE.

* *Comprobamos que se han almacenado datos en la tabla interna*

CHECK NOT (I_ZCLIENXX[] IS INITIAL).

* *En función de la opción seleccionada (Juego datos o call).*

```

IF NOT ( P_JDFL IS INITIAL ).
* Creamos el juego de datos con el contenido de la tabla I_ZCLIENXX
PERFORM CREAR_JUEGO_DATOS.
* Mostramos mensaje de 'Juego de datos creado'.
MESSAGE I000(38) WITH TEXT-003 P_JDATOS.
* En caso contrario se ha seleccionado Call
ELSE.
* Completamos la tabla interna I_BDCTAB
PERFORM COMPLETAR_BDCTAB.
* Llamamos a la transacción en modo sincrónico y no visualización
CALL TRANSACTION 'ZBXX' USING I_BDCTAB MODE 'N' UPDATE 'S'.
* Comprobamos el valor de retorno
IF ( SY-SUBRC = 0 ).
* Mostramos mensaje de datos almacenados
MESSAGE I000(38) WITH TEXT-004.
ENDIF.
ENDIF.
* Si hay error , mostramos el mensaje, con un mensaje de error
ELSE.
MESSAGE E000(38) WITH D_MERROR.
ENDIF.

*****
* Control de la pantalla de selección. *
*****

* Control de la salida de la pantalla de selección
AT SELECTION-SCREEN OUTPUT.
* Si esta seleccionada la opción de juego de datos
* hacemos que la entrada para el parámetro P_JDATOS sea obligatoria.
IF NOT ( P_JDFL IS INITIAL ).
* Recorremos todos los elementos de la pantalla.
LOOP AT SCREEN.
* Al parámetro P_JDATOS le hacemos obligatorio
CHECK SCREEN-NAME = 'P_JDATOS'.
SCREEN-REQUIRED = '1'.
MODIFY SCREEN.
ENDLOOP.
ENDIF.

*****
* Rutinas adicionales. *
*****
*&-----*
*& Form CREAR_JUEGO_DATOS
*&-----*
* Crea un juego de datos con el nombre indicado por el parámetro *
* P_JDATOS, para la transacción ZBXX con el contenido de la *

```

```

*   tabla interna I_ZCLIENXX.
*-----*
FORM CREAR_JUEGO_DATOS.

*****
* Proceso
*****

* Abrimos el juego de datos.
CALL FUNCTION 'BDC_OPEN_GROUP'
  EXPORTING
*   CLIENT      = SY-MANDT
*   DEST        = FILLER8
  GROUP        = P_JDATOS
*   HOLDDATE    = FILLER8
  KEEP         = ''
  USER         = SY-UNAME
*   RECORD      = FILLER1
* IMPORTING
*   QID         =
  EXCEPTIONS
    CLIENT_INVALID      = 1
    DESTINATION_INVALID = 2
    GROUP_INVALID       = 3
    GROUP_IS_LOCKED     = 4
    HOLDDATE_INVALID    = 5
    INTERNAL_ERROR      = 6
    QUEUE_ERROR         = 7
    RUNNING             = 8
    SYSTEM_LOCK_ERROR   = 9
    USER_INVALID        = 10
    OTHERS              = 11.

* Continuamos si no hay error
CHECK ( SY-SUBRC = 0 ).

* Completamos la tabla BDC de la forma oportuna
PERFORM COMPLETAR_BDCTAB.

* Una vez completado la transacción , la incluimos en el J.datos
CALL FUNCTION 'BDC_INSERT'
  EXPORTING
    TCODE      = 'ZBXX'
*   POST_LOCAL = NOVBLOCAL
*   PRINTING   = NOPRINT
  TABLES
    DYNPROTAB  = I_BDCTAB
  EXCEPTIONS
    INTERNAL_ERROR = 1
    NOT_OPEN      = 2

```

```

        QUEUE_ERROR    = 3
        TCODE_INVALID = 4
        PRINTING_INVALID = 5
        POSTING_INVALID = 6
        OTHERS        = 7.
* Comprobamos si hubo error, mostramos mensaje de error
IF ( SY-SUBRC <> 0 ).
    MESSAGE E000(38) WITH SY-SUBRC.
ENDIF.
* Una vez completado el Juego de datos , lo cerramos
CALL FUNCTION 'BDC_CLOSE_GROUP'
    EXCEPTIONS
        NOT_OPEN    = 1
        QUEUE_ERROR = 2
        OTHERS      = 3.
ENDFORM.          " CREAR_JUEGO_DATOS
*&-----*
*&   Form COMPLETAR_BDCTAB
*&-----*
*   Completa la tabla interna I_BDCTAB con los datos necesarios
* para llamar a la transacción ZBXX existentes en la tabla interna
* I_ZCLIENXX.
*-----*
FORM COMPLETAR_BDCTAB.
*****
* Definición de variables locales
*****

DATA: L_FNACI(10) TYPE C.          " Formateo de fecha para pantalla

*****
* Proceso
*****
* Inicializamos la tabla interna BDC
REFRESH I_BDCTAB.
* Completamos la tabla BDC de la forma oportuna
LOOP AT I_ZCLIENXX.
* Formateamos el campo fecha para salida (dd.mm.aaaa)
* en la pantalla no podemos indicar el formato interno (aaaammdd)
WRITE I_ZCLIENXX-FNACI TO L_FNACI.
* Completamos la pantalla inicial con los datos del cliente actual
PERFORM BDC_INSERT USING:
    'X' 'SAPMZBXX'      '9000',
    '' 'ZCLIENXX-BUKRS' I_ZCLIENXX-BUKRS,
    '' 'ZCLIENXX-NCLIE' I_ZCLIENXX-NCLIE,
    '' 'ZCLIENXX-NOMBR' I_ZCLIENXX-NOMBR,
    '' 'ZCLIENXX-APEL1' I_ZCLIENXX-APEL1,
    '' 'ZCLIENXX-APEL2' I_ZCLIENXX-APEL2,

```

```

'' 'ZCLIENXX-FNACI' L_FNACI,
'' 'D_PARTICULAR' 'X',
'' 'BDC_OKCODE' 'ENTE',
* En la pantalla inicial (con datos introducidos) damos alta
'X' 'SAPMZBXX' '9000',
'' 'BDC_OKCODE' 'ALTA'.
ENDLOOP.
* Introducimos el código de fin de transacción después del
* último cliente
PERFORM BDC_INSERT USING:
* En la pantalla inicial (abandonamos la transacción).
'X' 'SAPMZBXX' '9000',
'' 'BDC_OKCODE' '/BACK'.

ENDFORM. " COMPLETAR_BDCTAB

*&-----*
*& Form BDC_INSERT
*&-----*
* Inserta en la tabla interna global I_BDCTAB los campos recibido *
* Distinguiendo si se trata de un registro de dynpro o de un registro *
* de campo. *
*-----*
* -->PE_DYNBEGIN Flag de pantalla/ campo *
* -->PE_FNAM Nombre de programa / campo *
* -->PE_FVAL Valor pantalla / campo *
*-----*
FORM BDC_INSERT USING VALUE(PE_DYNBEGIN)
VALUE(PE_FNAM)
VALUE(PE_FVAL).

*****
* Proceso *
*****

* Inicializamos la cabecera de la tabla
CLEAR I_BDCTAB.
* Informamos el flag de pantalla
I_BDCTAB-DYNBEGIN = PE_DYNBEGIN.
* En función si es o no pantalla
IF ( PE_DYNBEGIN = 'X' ).
I_BDCTAB-PROGRAM = PE_FNAM.
I_BDCTAB-DYNPRO = PE_FVAL.
* Si se trata de un campo
ELSE.
I_BDCTAB-FNAM = PE_FNAM.
I_BDCTAB-FVAL = PE_FVAL.
ENDIF.
* Insertamos el registro en la tabla interna

```



```

* FECHA.      AUTOR.      DESCRIPCION MODIFICACION.      *
*****
REPORT ZINTE3X NO STANDARD PAGE HEADING.
*****
* Tablas del diccionario de datos                                     *
*****
TABLES: ZCLIENXX.          " Maestro de clientes.
*****
* Definición de tablas internas                                     *
*****
* Tabla interna para almacenar los registros del fichero
DATA: BEGIN OF I_ZCLIENXX OCCURS 0,
      BUKRS LIKE ZCLIENXX-BUKRS,    "Sociedad
      NCLIE LIKE ZCLIENXX-NCLIE,    "Número de cliente
      NOMBR LIKE ZCLIENXX-NOMBR,    "Nombre cliente
      APEL1 LIKE ZCLIENXX-APEL1,    "Primer apellido
      APEL2 LIKE ZCLIENXX-APEL2,    "Segundo apellido
      FNACI LIKE ZCLIENXX-FNACI,    "Fecha de nacimiento
      END OF I_ZCLIENXX.
*****
* Pantalla de selección                                           *
*****

* Parámetro nombre de fichero
SELECTION-SCREEN BEGIN OF BLOCK BLOQ1 WITH FRAME TITLE
TEXT-001.
* Parámetros
PARAMETERS:
* Sociedad
  P_BUKRS LIKE ZCLIENXX-BUKRS OBLIGATORY DEFAULT '0001',
* Nombre del fichero local (activamos minúsculas permitidas).
  P_NFICH(128) TYPE C OBLIGATORY DEFAULT 'C:\zcliexx.dat'
  LOWER CASE.
SELECTION-SCREEN END OF BLOCK BLOQ1.
*****
* Comienzo de selección                                           *
*****
START-OF-SELECTION.
* Seleccionamos los datos de la tabla
SELECT *
FROM ZCLIENXX INTO CORRESPONDING FIELDS OF TABLE I_ZCLIENXX
WHERE BUKRS = P_BUKRS.
* Comprobamos que se seleccionarán datos
CHECK ( SY-SUBRC = 0 ).
* Volcamos el contenido de la tabla interna al fichero local
CALL FUNCTION 'WS_DOWNLOAD'
EXPORTING

```

```

*      BIN_FILESIZE      = ''
*      CODEPAGE          = ''
      FILENAME          = P_NFICH
      FILETYPE          = 'ASC'
*      MODE              = ''
*      WK1_N_FORMAT     = ''
*      WK1_N_SIZE       = ''
*      WK1_T_FORMAT     = ''
*      WK1_T_SIZE       = ''
*      COL_SELECT       = ''
*      COL_SELECTMASK   = ''
*      NO_AUTH_CHECK    = ''
*      IMPORTING
*      FILELENGTH       =
TABLES
      DATA_TAB        = I_ZCLIENXX
*      FIELDNAMES       =
EXCEPTIONS
      FILE_OPEN_ERROR  = 1
      FILE_WRITE_ERROR = 2
      INVALID_FILESIZE = 3
      INVALID_TABLE_WIDTH = 4
      INVALID_TYPE     = 5
      NO_BATCH         = 6
      UNKNOWN_ERROR    = 7
      OTHERS           = 8.
*      Mostramos mensaje de fichero creado
      MESSAGE I000(38) WITH TEXT-001.

```

8.7 Legacy System Migration Workbench

Se ejecuta a través de la transacción LSMW. Esta herramienta permite realizar migraciones de datos de sistemas externos a R/3, fundamentalmente cargas de datos e interfaces.

Con la LSMW los datos se pueden cargar de 4 formas diferentes:

- Batch Input/Direct Input.
- Grabación de un Batch Input.
- BAPI
- IDOC's

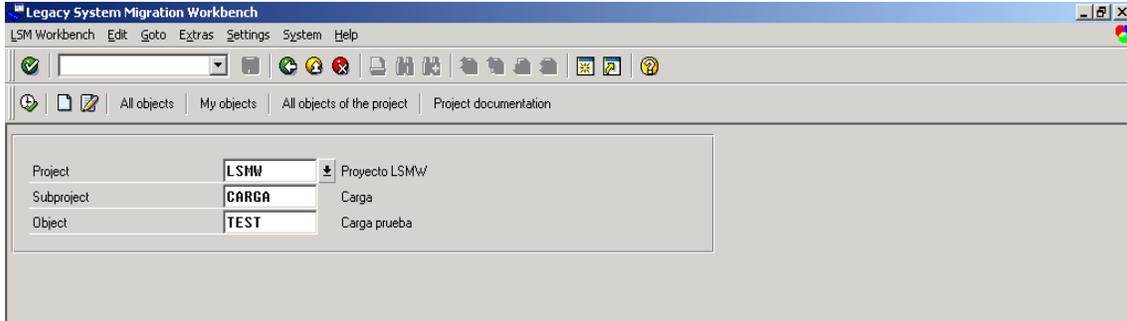
Para realizar una carga utilizando cualquier método de los anteriores, es necesario definir el tipo de objeto a cargar.

Al crear un nuevo Proyecto se deben introducir los siguientes datos:

- Proyecto (de tipo CHAR de 10 caracteres).

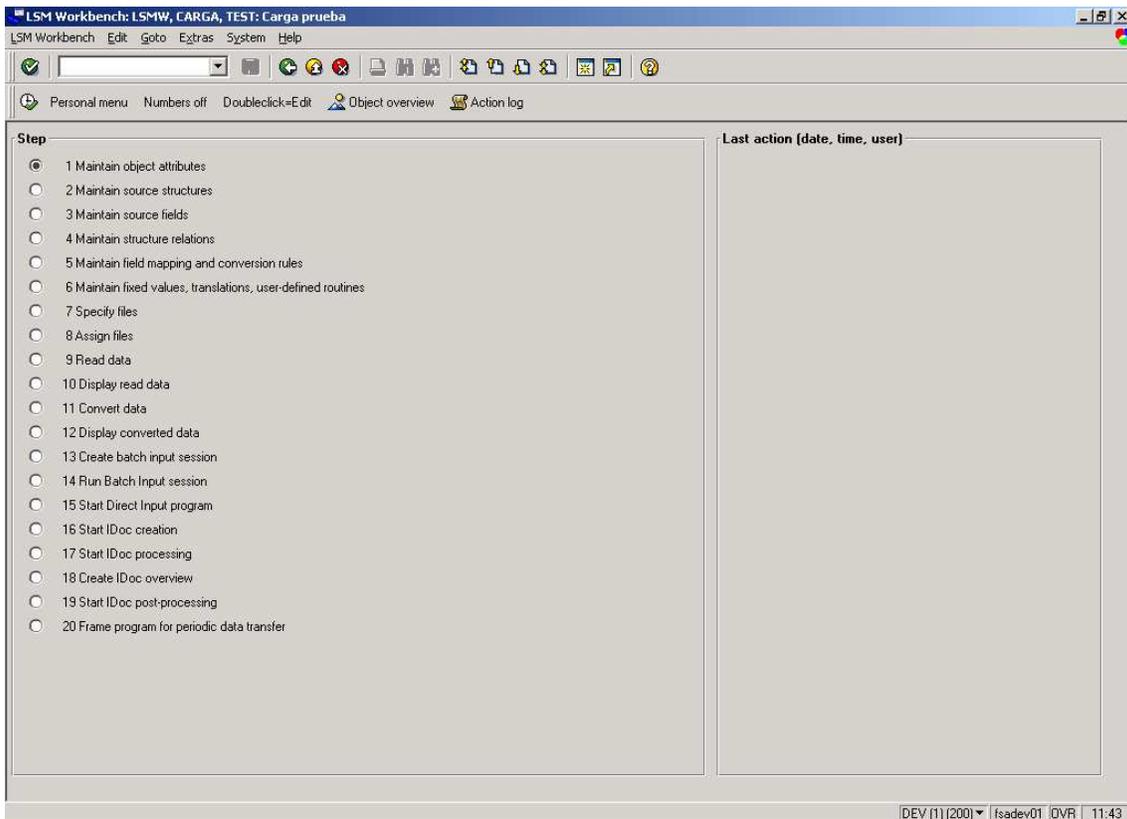
Curso programación ABAP IV

- Subproyecto (de tipo CHAR de 10 caracteres).
- Objeto (de tipo CHAR de 10 caracteres).



La primera vez que se entra en la transacción y antes de definir los datos que van a ser cargados, aparece una pantalla que indica los pasos que se deben definir.

Finalmente se muestran estos pasos se en la imagen siguiente:



9. Formularios

9.1 Introducción

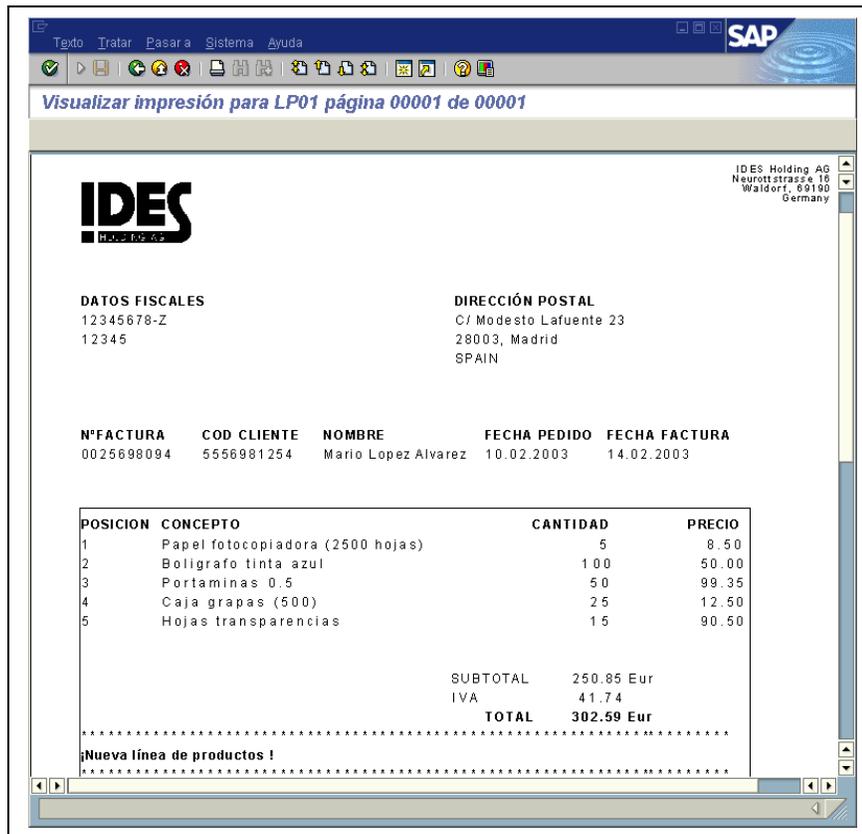
Las empresas necesitan imprimir habitualmente informes, facturas, pedidos, etc. con un formato predefinido. Estos documentos pueden contener textos que se mantienen constantes entre documentos, pero también puede haber datos variables. En muchas ocasiones la creación de estos documentos se produce en masa creando múltiples copias de un mismo tipo de documento pero con propiedades distintas. De esta forma, la creación de los documentos como facturas, pedidos etc. se puede realizar de forma automática. SAPscript resuelve todos estos problemas con una plataforma común.

La creación de los documentos implica a varios elementos. Desde el punto de vista del programador SAPscript se puede dividir en formularios y programas de control de formularios. En realidad están implicados también el diccionario de SAP y el composer (un elemento que crea el formulario a partir de los elementos y de los requisitos del programa de control) Los formularios especifican la apariencia del texto en el documento (forman la plantilla del documento) y el programa de control especifica que valores contendrán los campos del formulario. De esta forma, para cambiar la apariencia de un documento sólo es necesario cambiar el formulario. Por el contrario, si se desea cambiar el contenido del documento es necesario cambiar los elementos de texto así como el programa de control del formulario.

El programa de control del formulario controla la salida del documento hacia la impresora, fax, pantalla, número de copias... Este programa selecciona los datos que se van a mostrar en el formulario del diccionario de ABAP o de las entradas del usuario, así como el formulario controlando los textos que se deben imprimir, su secuencia y frecuencia.

La apariencia final del documento depende de la interacción del programa de control con el formulario. El programa de control inicializa y finaliza el proceso de impresión, transfiriendo los comandos de SAPscript al composer. El composer formatea el documento a partir de la información del diseño del formulario especificado en el programa de control. Si el documento contiene variables, el composer reemplaza el contenido de estas en su posición (fecha, hora...)

Ejemplo de una factura:

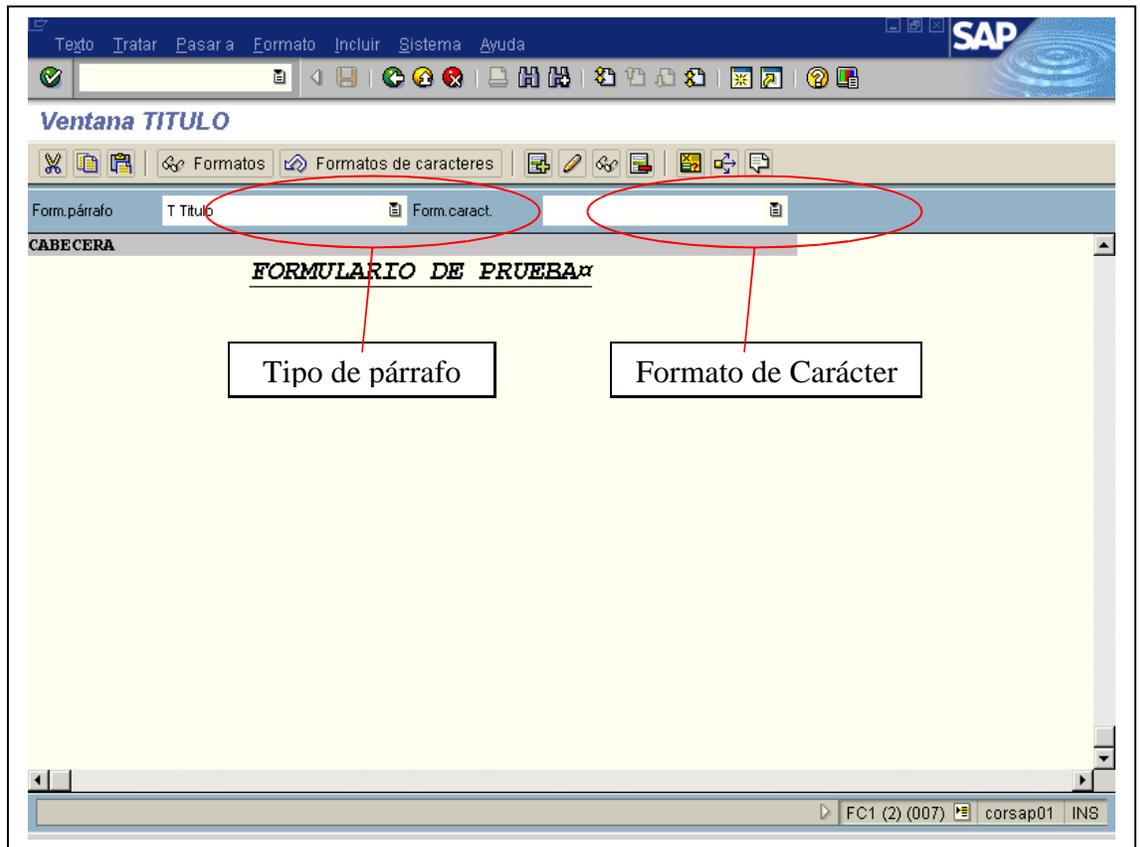


9.2 Editores de formularios

9.2.1 Editor Gráfico

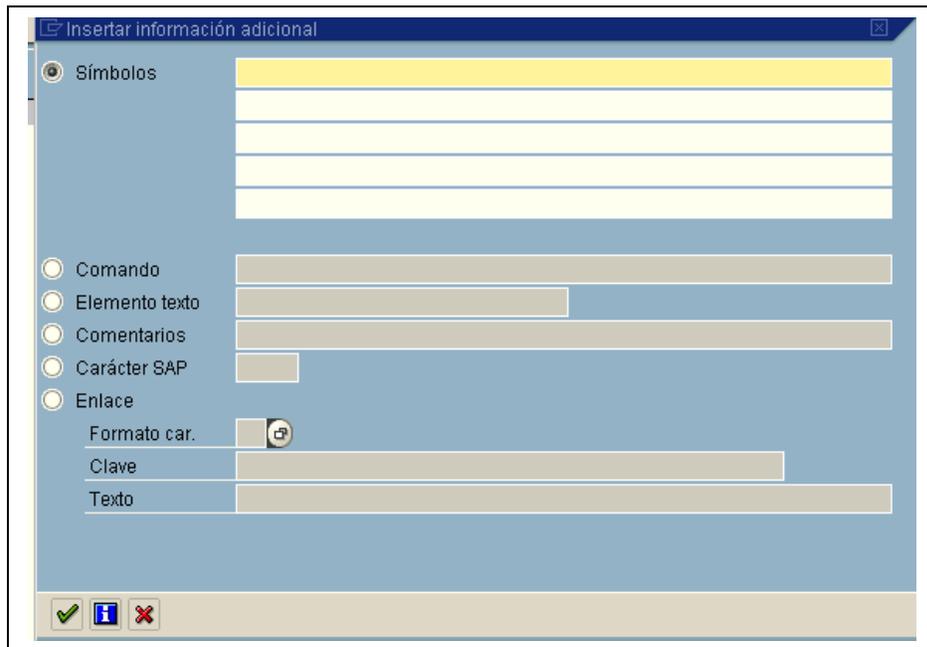
Si entramos al editor de texto, seleccionando primero una ventana y pulsando luego en Elementos de texto veremos la siguiente pantalla:

Nombre de la ventana en la que se define el elemento de texto.



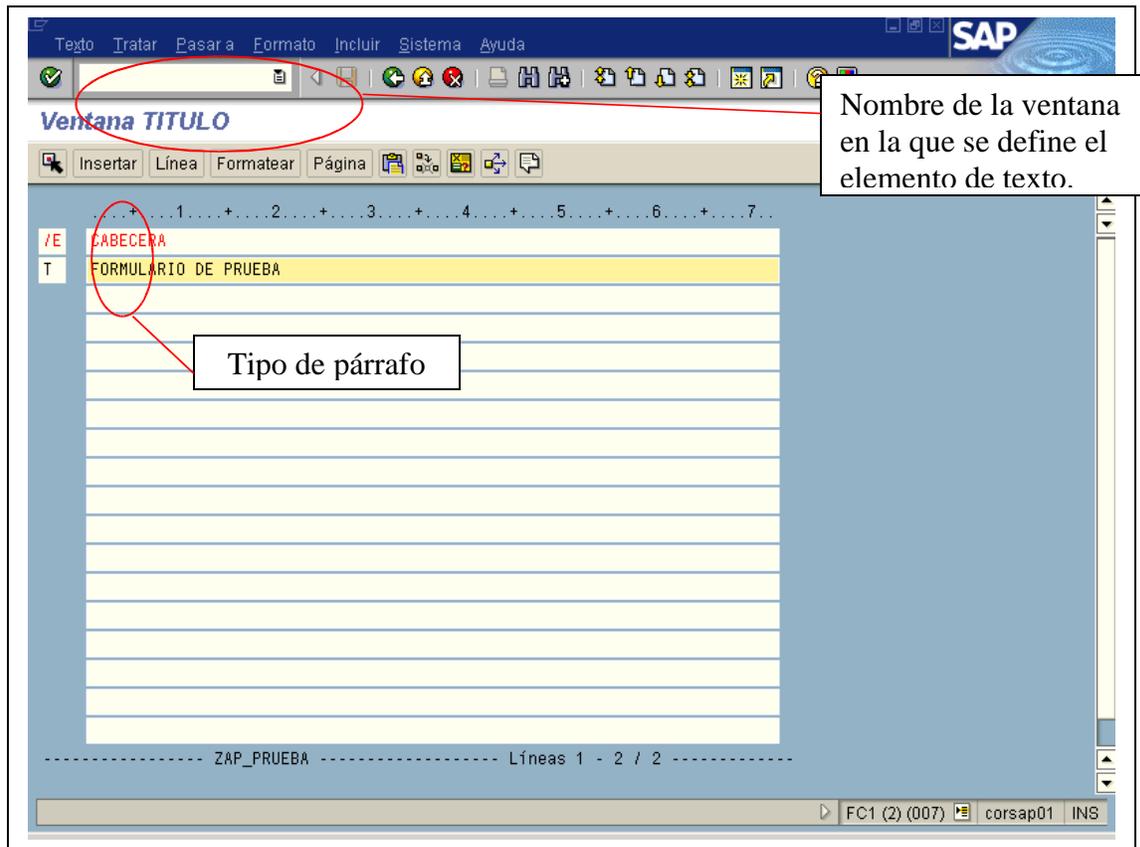
En el campo de Tipo de Párrafo podemos escoger entre los formato de párrafo que hemos definido en dicha ventana al igual que en el campo de Formato de Carácter podemos escoger entre los formatos de carácter que nos hemos definido.

Para introducir comandos, símbolos, elemento de textos, etc. dentro del formulario en el menú Tratar → Comando → Insertar Comando nos permite insertar dicha información adicional.

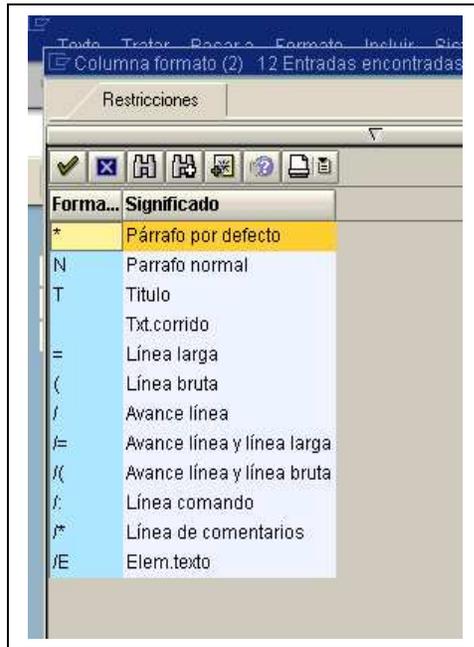


9.2.2 Editor Alfanumérico

Para pasar al editor de texto alfanumérico, desde el menú Pasar seleccionar la opción de Cambiar editor.

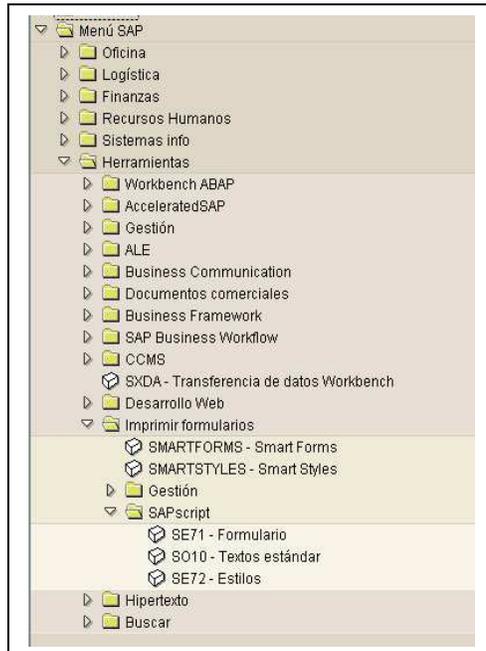


A diferencia del editor gráfico en el tipo de párrafo a parte de tener nuestro formato de párrafo definido anteriormente también podemos informar si la línea en cuestión es un comando, comentario, etc.

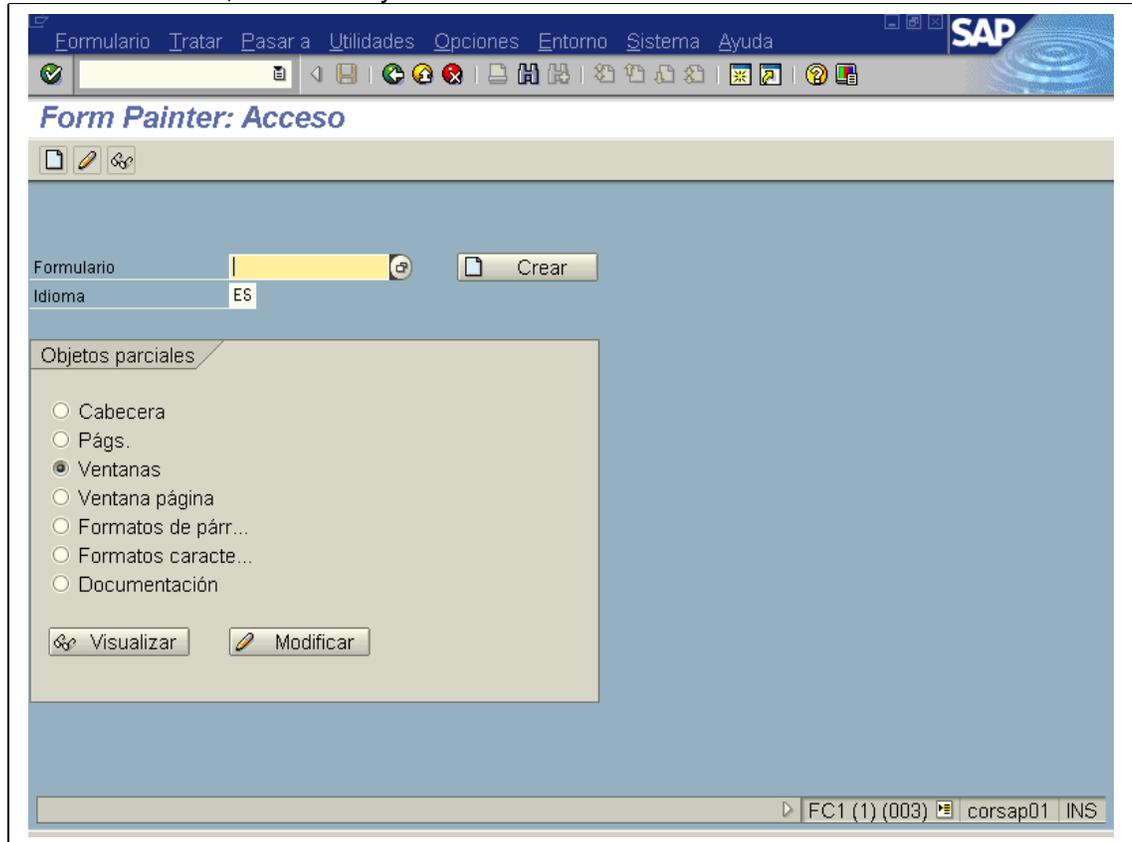


9.3 Estructura de un formulario

Para acceder al editor de formularios se sigue la ruta Herramientas -> Imprimir formularios->SAPscript-> Formulario, o directamente con la transacción SE71.



Accederemos a continuación a la pantalla Form Painter. Podemos crear un nuevo formulario, editar uno ya existente o modificarlo.



Nota: En el caso de no mostrar esta pantalla significará que estamos usando Form Painter gráfico. Es igual que el modo alfanumérico pero más visual. No obstante aquí seguiremos el modo alfanumérico accediendo por el menú->Opciones->Form Painter y desmarcando el flag de modo gráfico.

Un formulario especifica la disposición de los siguientes elementos en las páginas de un documento:

- Cabecera
- Páginas
- Ventanas
- Ventana página
- Formatos de párrafo
- Formatos caracteres
- Elementos de texto

9.3.1 Cabecera

La cabecera de un formulario consiste en atributos globales del formulario. Estos pueden ser *datos de gestión* (Nombre del formulario,

descripción, clase de desarrollo...) y *parametrizaciones básicas* (formato de página, fuente por defecto, párrafo por defecto...). Cuando se crea un formulario, se accede directamente a la cabecera, la cual sirve para definir los datos generales del formulario.

🕒 Datos de gestión

Formulario Tratar Pasara Atributos Utilidades Opciones Sistema Ayuda

Modificar cabecera de formulario: ZAP_PRUEBA

Páginas Ventanas Ventana página Formatos de párrafo Formatos de caracteres

Datos de gestión Parametrizaciones básicas

Info gestión

Formulario ZAP_PRUEBA

Significado Pueba de formulario

Status Activo - grabado

Clasificación

Clase de desarrollo \$TMP Temporary Objects (never transported!)

Nº mandante 003

Creado/a el 10.01.2003 10:17:16 De BAR10 Release 46C

Modificado el 10.01.2003 12:24:17 De BAR10 Release 46C

Atributos idioma

Clave de idioma ES

Idioma original ES

Traducir

A todos los idiomas

A un idioma

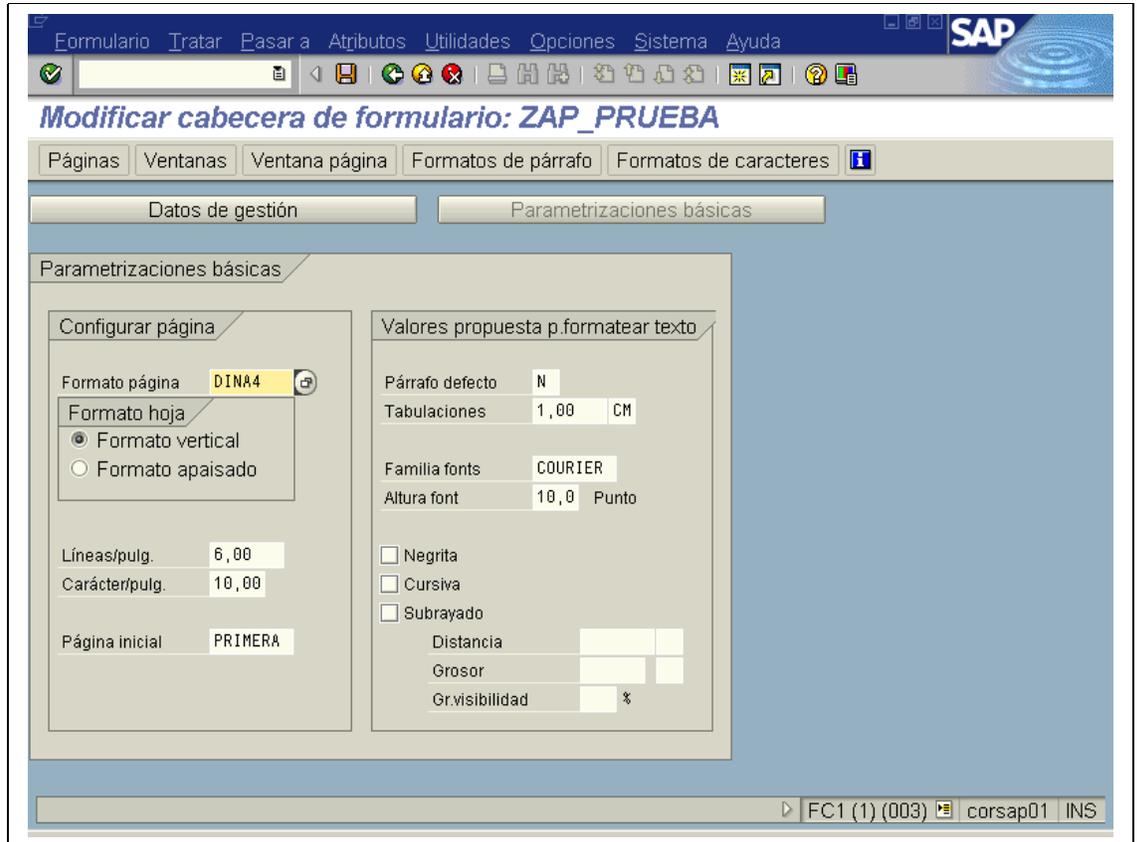
No traducir

FC1 (1) (003) corsap01 INS

Esta ventana está conformada por dos subdivisiones, info gestión y atributos idioma.

- **Info gestión:**
 - Formulario: El nombre de un formulario no deberá tener más de 16 posiciones y sólo deberán utilizarse letras mayúsculas y cifras, comenzando siempre por una letra.
 - Significado: Descripción breve del formulario
- **Atributos idioma:**
 - Se define el idioma principal y si el formulario podrá ser traducido a otros idiomas, o solo podrá mostrarse en el idioma original.

🕒 *Parametrizaciones básicas*



En la ventana de parametrizaciones básicas se distinguen dos divisiones, una conformada por la configuración de la página y la otra con los valores propuestos para el formato del texto

Configurar página:

Formato de página: Indica el tamaño del papel.

Formato hoja: Indica la orientación del papel.

Líneas/pulg.: Indica la cantidad de líneas de impresión por pulgada.

Carácter/pulg.: Indica la cantidad de caracteres de impresión por pulgada.

Página inicial: Primera página que se va a imprimir de nuestro formulario.

Valores propuesta p. formatear texto: Se especifican los valores tomados por defecto.

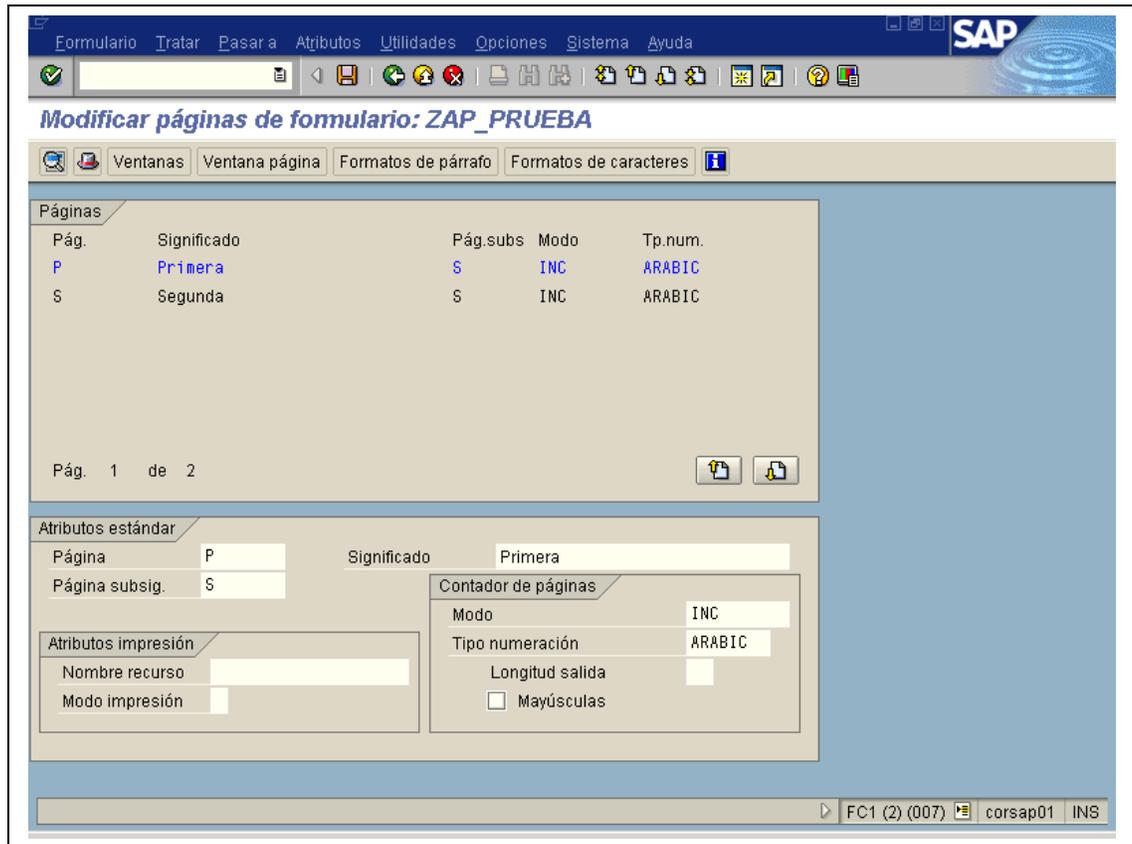
Párrafo por defecto: Si se indica '*' para el párrafo en el editor SAPscript, el sistema tomará el párrafo default para editar el texto.

Tabulaciones: Distancia entre las tabulaciones en un formulario.

Familia fonts, altura font...: Definen el formato de carácter por defecto.

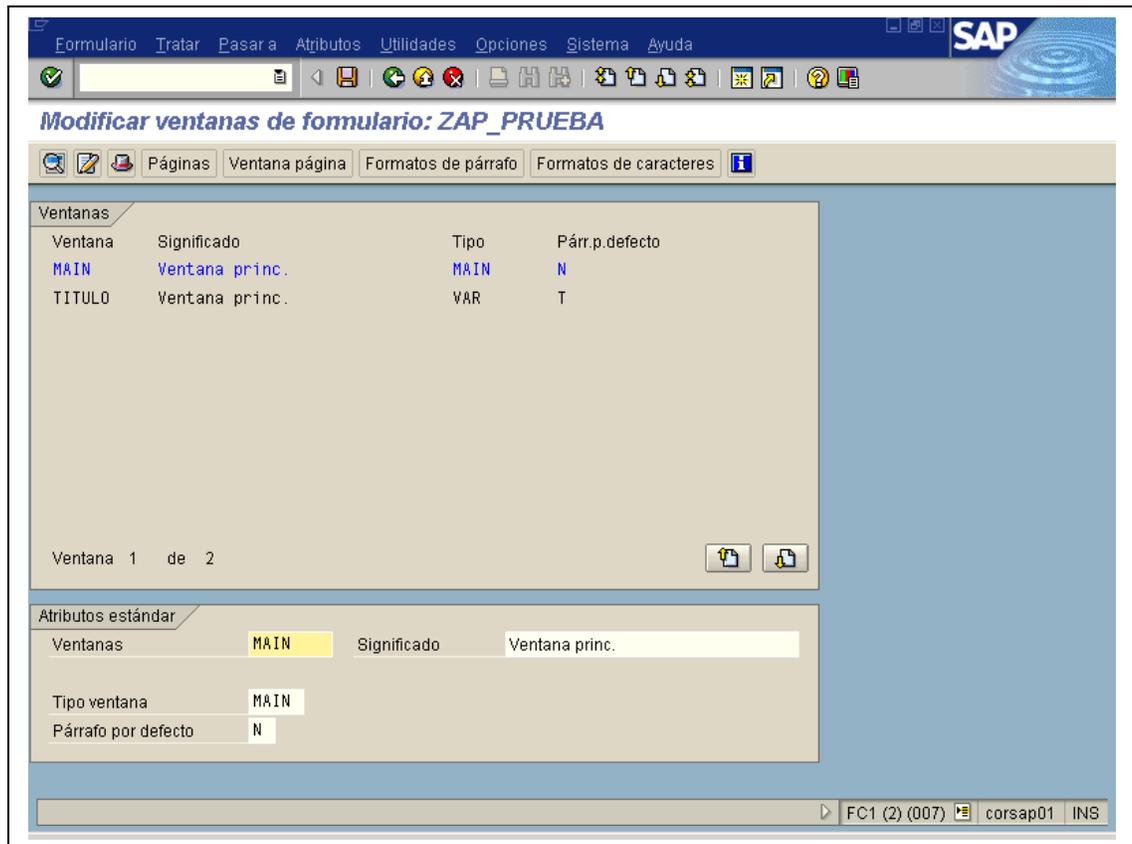
9.3.2 Páginas

Representan las distintas páginas del documento. Estas suelen tener distinto aspecto unas de otras, la primera página de un fax contiene información distinta a la que se mostrara en las páginas siguientes. Para cada formulario se ha de definir al menos una página. En este apartado daremos una descripción a la página, indicaremos cual es la página siguiente y cómo se actualiza el contador de páginas, así como los atributos de impresión como son el modo de impresión (Por defecto, SYMPLEX, DUPLEX o TRIPLEX) y el nombre del recurso que será la bandeja de la impresora de donde tomará el papel (TRY01, TRY02, TRY03).



9.3.3 Ventanas

Las ventanas representan áreas que se posicionarán sobre las páginas. En ellas pondremos el párrafo por defecto, el nombre de la ventana, su significado y el tipo de ventana.



Hay 4 tipos de ventanas:

MAIN: Es la ventana principal en la que se escribirá el texto variable, como podría ser el cuerpo de una carta. Esta ventana puede extenderse a más de una página.

VAR: Es una ventana cuyo contenido puede variar. Estas ventanas deben definirse en cada página.

CONST: Define una ventana cuyo contenido no cambia.

GRAPH: Define una ventana con una imagen gráfica.

9.3.4 Ventana página

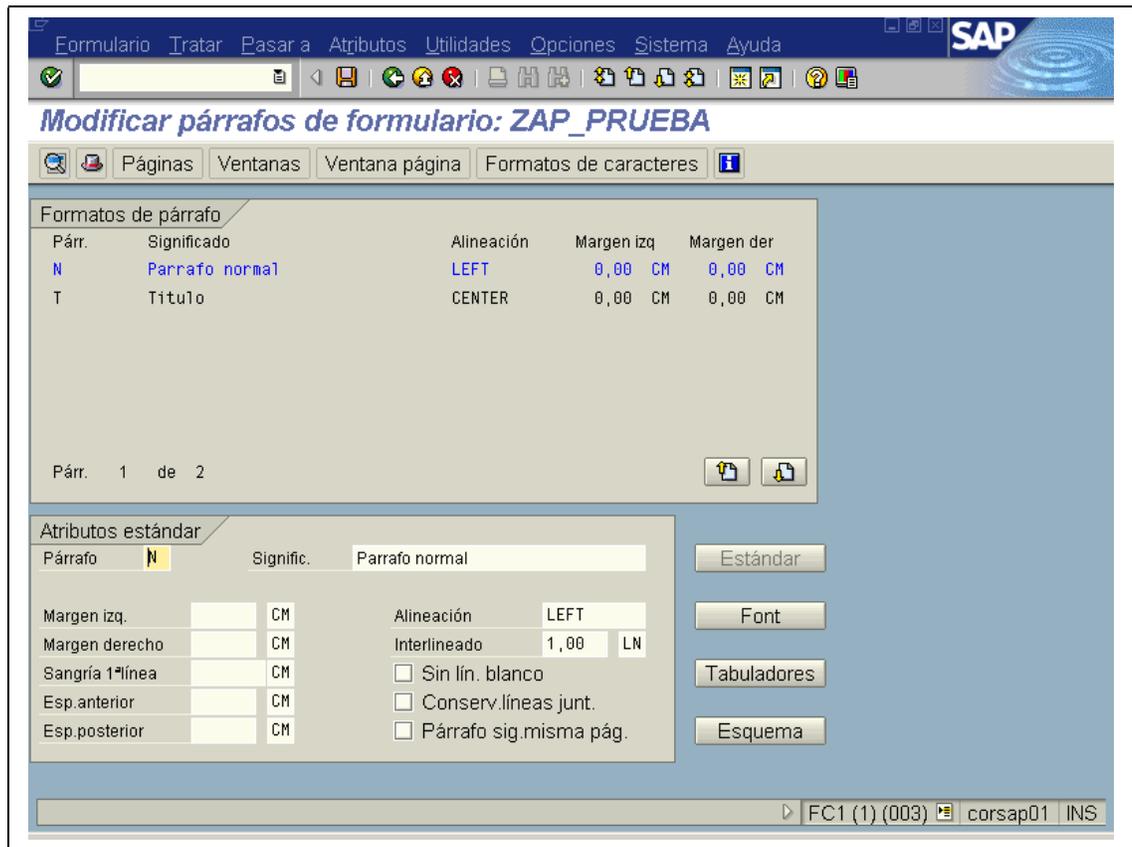
En este apartado, se especifica la posición y tamaño de las ventanas en cada página. Para añadir página, se realiza por medio del botón derecho del ratón en la opción crear elemento.



9.3.5 Formatos de párrafos

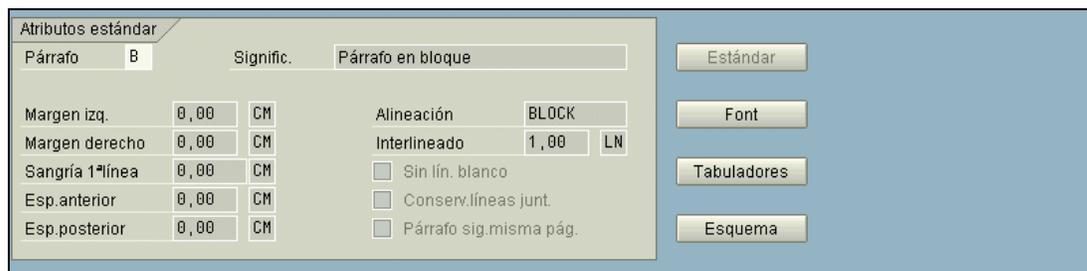
Los formatos de párrafos definen las características de estos, esta pantalla esta dividida en dos partes. En la zona superior se muestra una lista con todos los formatos de párrafo para el formulario, observándose las características generales de cada párrafo.

Los atributos de cada párrafo están divididos en 4 partes:



Para la creación de un nuevo párrafo se añade en la casilla de párrafo en la zona de atributos se añade el nombre que le identificara con uno o dos caracteres

🕒 *Datos estándar*



Se define las características propias del párrafo, como márgenes, sangría y alineación.

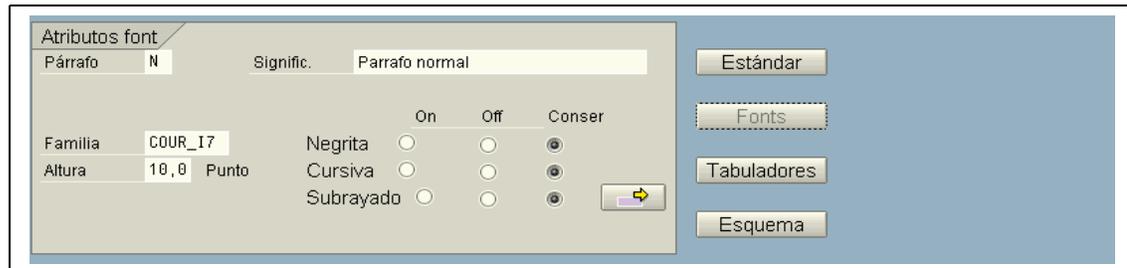
Párrafo: Define el nombre con el cual se identificara con posterioridad las distintas características del párrafo creado.

Significado: Breve descripción identificativa del párrafo.

Margen izquierdo y derecho: Distancia con la ventana que contenga el párrafo.
Sangría 1ª línea: Sangría.
Esp. anterior / posterior: Espacio de comienzo con respecto al párrafo anterior / posterior.
Alineación: Alineación del párrafo.
Interlineado: Espacio entre cada línea.

⌚ *Datos de Fuentes*

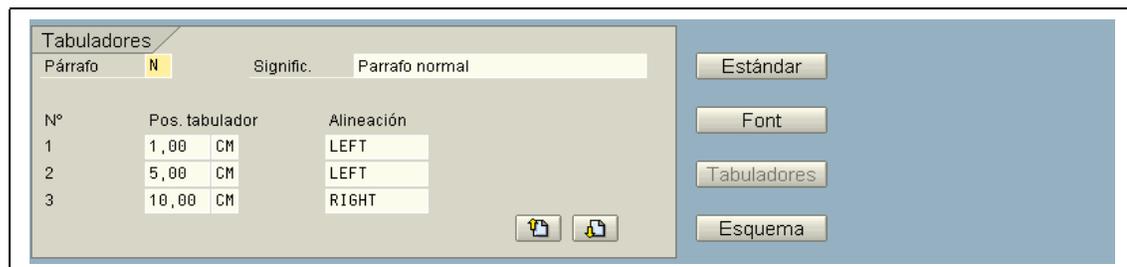
Contiene las características de la fuente así como su formato.



Familia: fuente que se usará en el párrafo.
Altura: Tamaño de la fuente a utilizar
Negrita, cursiva y subrayado: Como indica el título permite la opción de que el párrafo este formateado con alguna de estas características si se marca la opción "On" quedarán activadas, si se selecciona la opción "Conser" se usará el valor que tenía el texto anterior.

⌚ *Datos de tabuladores*

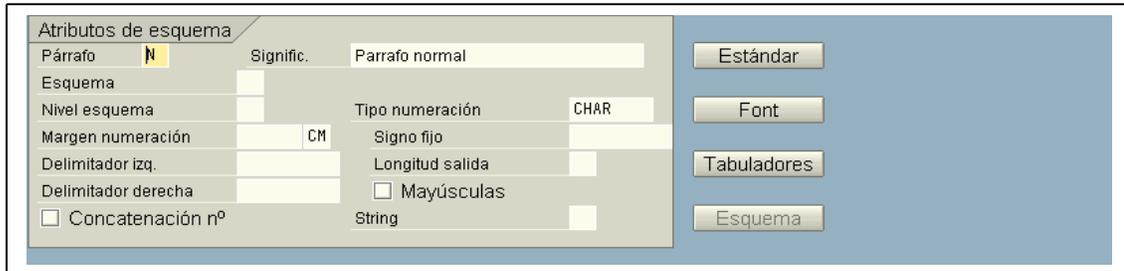
Aquí se definen las distintas posiciones de tabulación que necesitemos para cada párrafo.



Le deberemos indicar una posición, ya sea en centímetros (CM), caracteres (CH), milímetros (MM), puntos (PT) y un tipo de alineación: izquierda (LEFT), derecha (RIGHT), centrado (CENTER), al signo (SIGN) o a la coma decimal (DECIMAL)

⌚ *Datos de esquema*

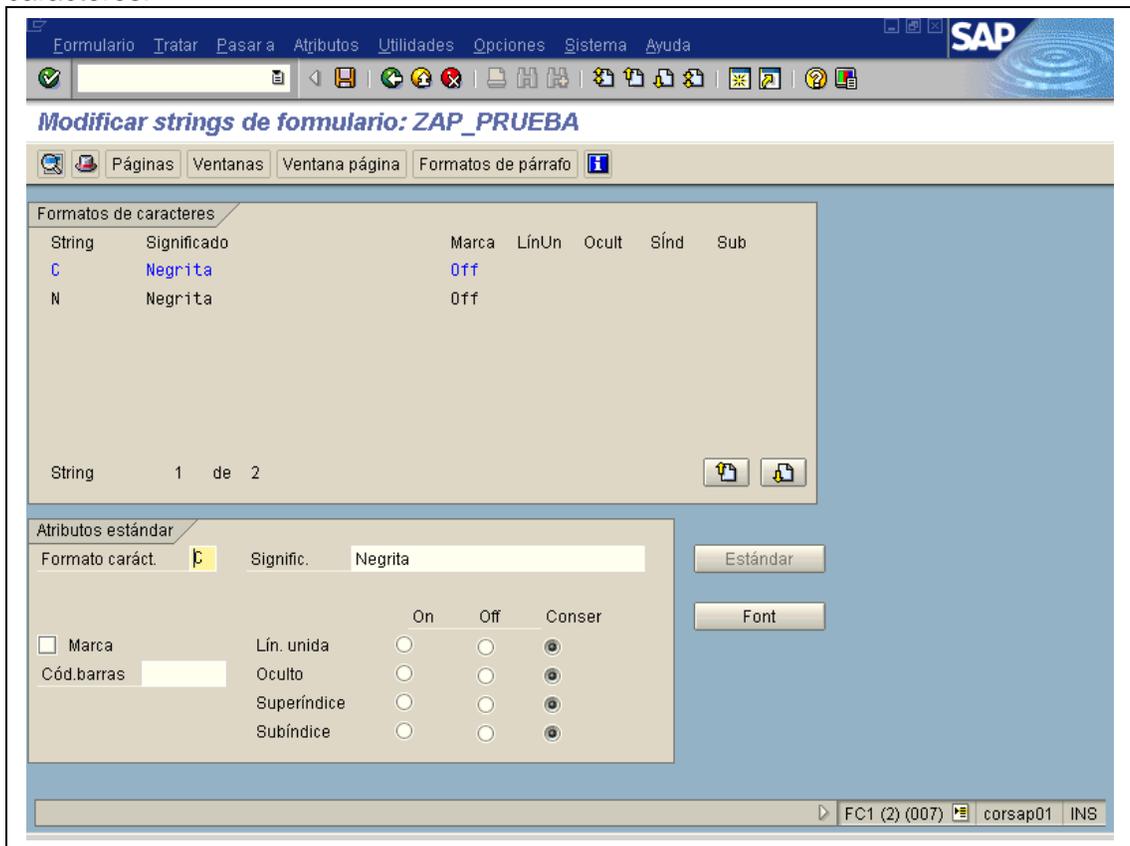
Es posible definir numeración y marcas automáticas de forma que podamos estructurar texto en capítulos, subcapítulos y secciones.



El signo fijo será el carácter que se antepondrá siempre al párrafo, si el signo es '_' está se convertirá en espacio en blanco.

9.3.6 Formatos de caracteres

Dentro de cualquier párrafo es posible cambiar el tipo de letra para uno o más caracteres.



⌚ Atributos estándar

Formato caráct.: Nombre con un máximo de dos letras con el cual se identificara el tipo de formato de carácter creado.

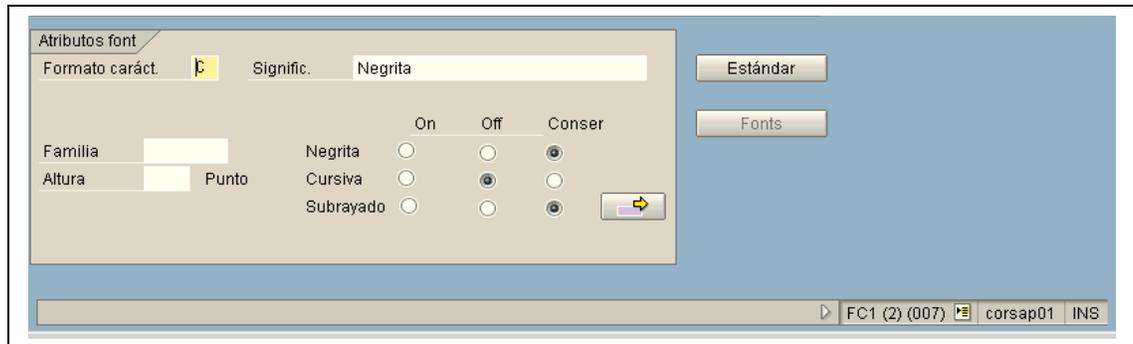
Signific.: Breve descripción que defina el formato de caracteres.

Cód. Barras: Se introduce el código numérico de barras para la posterior visualización del mismo.

Lín. Unida, oculto, superíndice, subíndice: Se especifican las distintas opciones que tendrá el formato de caracteres. "Conser" conservara los parámetros especificados en el texto anterior.

⌚ *Atributos font*

Contiene los datos de la fuente que se usara en el formato de carácter.



Las características en los atributos Font son los mismos que para los formatos de párrafo.

9.3.7 Elementos de texto

Los elementos de texto son componentes individuales de una ventana. Puede haber más de un elemento de texto por ventana y se distinguen dos tipos. Los elementos de texto pueden contener variables e instrucciones de control SAPscript. Para acceder a ellos nos situamos en la pantalla de Ventana Pagina, seleccionamos la ventana donde queremos insertar los elementos de texto y pulsamos el botón de Elementos de texto (F9).



El programa de impresión accede a los elementos de texto por nombre, los formatea e imprime en la ventana correspondiente.

9.4 SAPScript

9.4.1 Cajas, líneas y sombreados

Se pueden dibujar cajas y líneas en SAPScript mediante los siguientes comandos:

BOX: Dibuja una caja o una línea.

POSITION: Especifica el punto inicial de una caja o línea

SIZE: Especifica la anchura o altura de una caja.

Ejemplo:

Creamos una nueva ventana de tipo CONST de idénticas proporciones que la ventana MAIN que dibuje una caja que rodee las dos columnas de la ventana MAIN del ejemplo.

/: BOX XPOS '0.5' CM WIDTH 8 CM HEIGHT 15 CM FRAME 10 TW.

/: BOX XPOS '8.5' CM WIDTH 8 CM HEIGHT 15 CM FRAME 10 TW.

9.4.2 Comandos de control

SAPScript dispone de una serie de comandos que permiten obtener un control total sobre el texto. Estos comandos son introducidos en el editor de textos igual que una línea normal excepto que el deberemos seleccionar el párrafo /:

NEW-PAGE: Provoca el salto automático de página

PROTECT – ENDPROTECT: Se asegura que el texto introducido entre estos dos comandos aparezca siempre en una misma página.

NEW-WINDOW: Llama automáticamente a la siguiente ventana MAIN definida en una misma página.

DEFINE: Permite crear una constante con un valor dado.

SET DATE MASK: Define el formato de los campos fecha.

SET TIME MASK: Define el formato de los campos hora.

SET COUNTRY: Define el formato de ciertos campos como el punto para los millares adaptados a un país específico.

SET SIGN: Indica la posición del signo.

RESET: Inicializa el contador de un párrafo con numeración.

INCLUDE: Inserta el contenido de otro texto en el texto actual.

STYLE: Cambia el contenido del estilo actual del texto.

ADDRESS – ENDADDRESS: Formatea una dirección de acuerdo con las convenciones postales del país definido por el parámetro COUNTRY.

TOP – ENDTOP: Especifica líneas de texto que siempre aparecerán en la parte superior de la ventana MAIN.

BOTTOM – ENDBOTTOM: Especifica líneas de texto que aparecerán en la parte inferior de la ventana MAIN.

IF – ENDIF: Permite especificar que líneas debería imprimirse cuando se cumplan ciertas condiciones.

CASE: Cubre el caso de múltiples comandos IF anidados.

PERFORM: Permite llamar una rutina de un programa ABAP

PRINT-CONTROL: Llama directamente ciertas funciones de la impresora.

BOX, POSITION, LINE: Dibuja líneas y cajas.

HEX – ENDHEX: Envía a la impresora órdenes en el lenguaje que utiliza la impresora.

SUMMING: Acumula un valor total para un símbolo de programa.

9.4.3 Símbolos SAPScript

La información variable se introduce en los formularios SAPScript mediante "símbolos" o variables que SAP reconoce porque van rodeadas de '&'. Ej.: &symbol&.

Símbolos del sistema: variable como fecha, hora...

Símbolos de programa: variables almacenas en aplicaciones SAP como campos del diccionario de datos o variables globales de los programas.

Símbolos estándar definidos en la tabla TTDTG: El valor de estos símbolos es dependiente del lenguaje y puede contener hasta 60 caracteres. SAP mantiene esta tabla con valores estándar.

Símbolos de texto: Aquellos que no corresponden a los tipos de símbolos definidos anteriormente. Son definidos por el usuario en el editor de texto, eligiendo Incluir -> Símbolos -> Texto. O bien definiéndolos con el comando INCLUDE.

9.4.4 Símbolos del sistema

&DATE&	Fecha del sistema. Se imprimirá de acuerdo al tipo de SET DATE MASK definida anteriormente.
&DAY&	Día del sistema.
&MONTH&	Mes
&YEAR&	Año
&NAME_OF_DAY&	Nombre del día de la semana del recogido en el SET DATE MASK.
&NAME_OF_MONTH&	Nombre del mes
&TIME&	Hora según el formato especificado en el SET TIME MASK.
&HOURS&	Hora
&MINUTES&	Minuto
&SECONDS&	Segundo
&PAGE&	Número de la página actual
&NEXTPAGE&	Número de la página siguiente a la actual
&DEVICE&	Tipo del dispositivo de salida (PRINTER, SCREEN, TELEX, ABAP).
&SPACE&	Espacio.
&ULINE&	Línea de subrayado
&VLIN&	Línea vertical

9.4.5 Campos generales de SAPScript

&SAPSCRIPT-SUBRC& Recibe un valore después de ejecutar un comando INCLUDE. (0 si lo encuentra, 4 si no).

&SAPSCRIPT-DRIVER& Nombre del dispositivo de salida (POST, HPL2, PRES).

&SAPSCRIPT-FORMPAGES& N° total de páginas del form.

&SAPSCRIPT-JOBPAGES& N° total de páginas de todos los formularios contenido en la petición de impresión actual.
&SAPSCRIPT_CONTER_X& (x = 0...9) Representan 10 variables de contador que se puede utilizar en el texto y formularios para cualquier propósito.

9.4.6 Opciones de formato de los símbolos

Desplazamiento	&symbol+offset&
Longitud de salida	&symbol(length)&
Omisión del signo	&symbol(S)&
Mostrar el signo por la izquierda	&symbol(<)&
Mostrar el signo por la derecha	&symbol(>)&
Omitir los ceros iniciales	&symbol(Z)&
Comprimir los espacios	&symbol(C)&
Número de decimales	&symbol(.2)&
Omitir el indicador de miles	&symbol(T)&
Especificar exponente	&symbol(E2)&
Alinear a la derecha	&symbol(8R)&
Rellenar de caracteres	&symbol(Ff)&
Suprimir valores iniciales	&symbol(l)&
Ignorar rutinas de conversión	&symbol(K)&
Cambiar valor de un contador	&SAPSCRIPT_COUNTER_X(+)&
Textos precedentes	&'pre-text'symbol'post-text'&

9.4.7 Formularios en varios idiomas

Una vez tenemos creado el formulario en un idioma padre, podemos crearlo en otros idiomas. Para ello en la pantalla de mantenimiento de formularios, seleccionamos el formulario que acabamos de crear, seleccionamos el nuevo lenguaje del formulario y seleccionamos crear. Veremos que trabajamos con una copia del formulario anterior en el que lo único que deberemos hacer será traducir los textos y adaptar el formulario, si fuera necesario, a las peculiaridades del nuevo idioma.

Para llamar al formulario en distintos idiomas se realiza mediante la función de abrir formulario, indicando el idioma requerido. Si el formulario no existiera en ese idioma, se abrirá el formulario en el idioma padre.

```
CALL FUNCTION 'OPEN_FORM'  
  EXPORTING  
    FORM      = 'Z_PRUEBA'  
    LANGUAGE  = P_IDIOMA  
    OPTIONS   = ITCPO  
    DEVICE    = 'PRINTER'  
    DIALOG    = 'X'  
  EXCEPTIONS  
    OTHERS    = 1.
```

9.4.8 Inclusión de gráficos

Para incluir gráficos en un formulario, primero este debe estar grabado en SAP en formato TIFF, BMP o en forma de elemento de texto. Luego se incluye en la ventana deseada a través del menú Incluir -> Función gráfica y seleccionando el gráfico. Esto generará una línea de comando como la siguiente:

```
BITMAP IDES_LOGO OBJECT GRAPHICS ID BMAP TYPE BMON DPI 300
```

9.5 Programa de impresión del formulario

Una vez finalizado el diseño del formulario es necesario crear un programa que gestione la impresión del mismo. Para ello hay una serie de funciones Standard de SAP que gestionan todos los parámetros de salida. De todas ellas, las más relevantes son:

OPEN_FORM
WRITE_FORM
CLOSE_FORM

OPEN_FORM: Este modulo de función abre un formulario para su impresión. Esta función ha de ejecutarse antes que cualquier otra función que actúe sobre el formulario (WRITE_FORM, START_FORM, CONTROL_FORM...). Cada vez que se utiliza la función OPEN_FORM es necesario cerrar el formulario (CLOSE_FORM) para que este se imprima. Dentro de un mismo programa puede haber varios pares de llamadas a las funciones OPEN_FORM y CLOSE_FORM.

CLOSE_FORM: Cierra un formulario abierto previamente con la función OPEN_FORM.

WRITE_FORM: El sistema muestra un elemento de texto determinado del formulario. El elemento de texto se especifica en el parámetro exportado ELEMENT. En el parámetro WINDOW se puede especificar el nombre de la ventana de salida.

Estructura ITCPD representa los parámetros de control del formato de salida. Esta estructura se puede utilizar en los módulos de función PRINT_TEXT y OPEN_FORM en el parámetro OPTIONS.

TDPAGESLCT	SAPscript: seleccionar página de impresión
TDPREVIEW	SAPscript: habilitar vista previa
TDNOPREV	SAPscript: deshabilitar vista previa
TDNOPRINT	SAPscript: deshabilitar impresión desde vista previa
TDTITLE	SAPscript: Título de la pantalla de selección
TDPROGRAM	SAPscript: nombre del programa de símbolos de sustitución

TDTEST	SAPscript: visualización previa
TDIEXIT	SAPscript: volver inmediatamente después de la impresión
TDGETOTF	SAPscript: valor de retorno de la tabla OTF, no hay impresión
TDSCRNPOS	SAPscript: posición del OTF en la pantalla
TDDEST	Spool: nombre del dispositivo de salida
TDPRINTE	Spool: nombre del tipo de dispositivo
TDCOPIES	Spool: número de copias
TDNEWID	Spool: nueva petición
TDIMMED	Spool: petición de impresión inmediata
TDDELETE	Spool: borrar petición después de la impresión
TDLIFETIME	Spool: tiempo de retención de la petición
TDDATASET	Spool: identificación de la petición
TDSUFFIX1	Spool: primer sufijo de la petición
TDSUFFIX2	Spool: segundo sufijo de la petición
TDAUTHORITY	Spool: autorización para la petición
TDARMOD	Spool: modo de archivo
TDCOVER	Spool: imprimir portada
TDCOVTITLE	Spool: portada: título
TDRECEIVER	Spool: portada: nombre del destinatario
TDDIVISION	Spool: portada: nombre de la división
TDSCHEDULE	SAPcomm: tipo del tiempo de envío estimado
TDSENDDATE	SAPcomm: fecha de envío solicitada
TDSENDTIME	SAPcomm: hora de envío solicitada
TDTELELAND	SAPcomm: código del país destinatario
TDTELENUM	SAPcomm: número de marcación

Ejemplo de formulario

Formulario

 Formulario ZPRUEBA

Significado Formulario de prueba

Atributos std.
 Página inicial PRICIPAL
 Párrafo defecto DF
 Tabulaciones 1,00 CM
 Formato página DINA4
 Formato hoja Formato vertical
 Líneas/pulg. 6,00
 Carácter/pulg. 10,00

Atributos font
Familia fonts COURIER
Altura font 12,0 Punto
Negrita no
Cursiva no
Subrayado no

Carácteres Atributos

DF Normal
Atributos std.
Marca no
Atributos font
Familia fonts HELVE
Altura font 10,0 Punto

NG Negrita
Atributos std.
Marca no
Atributos font
Familia fonts HELVE
Altura font 10,0 Punto
Negrita sí

Párrafos Atributos

AD Dirección propia
Atributos std.
Interlineado 0.50 LN
Alineación Alin.derecha
Atributos font
Familia fonts HELVE
Altura font 8,0 Punto

DF Párrafo por defecto
Atributos std.
Interlineado 1.00 LN
Alineación alin.izq.
Atributos font
Familia fonts HELVE
Altura font 10,0 Punto

NG Negrita

Atributos std.
Interlineado 1.00 LN
Alineación alin.izq.
Atributos font
Familia fonts HELVE
Altura font 10,0 Punto
Negrita sí

Ventanas Atributos

ADDRESS Dirección
Tipo ventana VAR
Párr.p.defecto DF

CONTROL Ventana de control
Tipo ventana VAR
Párr.p.defecto DF

D_FISCAL Datos fiscales
Tipo ventana VAR
Párr.p.defecto DF

FACTURA Datos Factura
Tipo ventana VAR
Párr.p.defecto DF

LOGO Ventana de logo
Tipo ventana CONST
Párr.p.defecto DF

MAIN Ventana princ.
Tipo ventana MAIN
Párr.p.defecto DF

MARCO marco de la ventana main
Tipo ventana CONST
Párr.p.defecto DF

MY_ADD Dirección propia
Tipo ventana CONST
Párr.p.defecto AD

PAGO Forma de pago
Tipo ventana VAR
Párr.p.defecto DF

Páginas Atributos

PRICIPAL Primera página
Atributos std.
 Pág.subsiguiente SEGUNDA
Atributos impresión
 Modo impresión S
Contad.pág.
 Modo INC
 Tipo numeración cifras árabes
Ventana página
MAIN Margen izqdo. 1.25 CM
 Margen superior 8.89 CM
 Ancho ventana 18.27 CM
 Altura ventana 12.47 CM
ADDRESS Margen izqdo. 10.49 CM
 Margen superior 3.95 CM
 Ancho ventana 9.14 CM
 Altura ventana 1.98 CM
CONTROL Margen izqdo. 16.50 CM
 Margen superior 22.00 CM
 Ancho ventana 3.25 CM
 Altura ventana 3.00 CM
D_FISCAL Margen izqdo. 1.25 CM
 Margen superior 3.95 CM
 Ancho ventana 8.89 CM
 Altura ventana 1.98 CM
FACTURA Margen izqdo. 1.25 CM
 Margen superior 6.91 CM
 Ancho ventana 18.40 CM
 Altura ventana 1.48 CM
LOGO Margen izqdo. 1.25 CM
 Margen superior 1.25 CM
 Ancho ventana 1.25 CM
 Altura ventana 1.25 CM
MARCO Margen izqdo. 1.20 CM
 Margen superior 8.84 CM
 Ancho ventana 18.27 CM
 Altura ventana 12.47 CM
MY_ADD Margen izqdo. 10.50 CM
 Margen superior 1.25 CM
 Ancho ventana 9.25 CM
 Altura ventana 2.75 CM
PAGO Margen izqdo. 1.25 CM
 Margen superior 22.00 CM
 Ancho ventana 14.70 CM

Altura ventana 3.00 CM

SEGUNDA Página segunda y siguientes
Atributos std.
Pág.subsiguiente SEGUNDA
Contad.pág.
Modo INC
Tipo numeración cifras árabes
Ventana página
MAIN Margen izqdo. 1.25 CM
Margen superior 11.50 CM
Ancho ventana 18.27 CM
Altura ventana 10.00 CM

Elementos texto para ventanas:

ADDRESS

* <ng>DIRECCIÓN POSTAL</>
/: ADDRESS
* &CALLE&
* &CIUDAD&
* &PAIS&
/: ENDADDRESS
*

CONTROL

* <ng>CONTROL</>
*

D_FISCAL

* <ng>DATOS FISCALES</>
* &C_NIF&
* &N_RESERVA&

FACTURA

NG NºFACTURA,,COD CLIENTE,,NOMBRE,, , , ,FECHA PEDIDO,,FECHA
FACTURA
DF &CABECERA-NUMERO_FACTURA&,,&CABECERA-
IDCLIENTE&,,&CABECERA-DESCRIPCION&
 ,,&cabecera-fecha_pedido&,,&Cabecera-fecha_factura&

LOGO

*
/: BITMAP IDES_LOGO OBJECT GRAPHICS ID BMAP TYPE BMON DPI 300
*

MAIN
Elemento ITEM_HEADER
NG POSICION,,CONCEPTO,, , , , , , , , , , CANTIDAD,, ,PRECIO

Elemento DATOS
DF &i_posicion-posicion&,, ,&i_posicion-concepto(50)&,,
 &i_posicion-cantidad&,, , &I_POSICION-PRECIO(5)&

Elemento PUBLICIDAD
/: PROTECT
NG *****
NG &C_MENSAJE&
NG *****
/: ENDPROTECT

Elemento TOTAL
DF
DF
DF ,, , , , , , , , , , SUBTOTAL,, &SUBTOTAL&
DF ,, , , , , , , , , , IVA,, , , , &IVA&
NG ,, , , , , , , , , , TOTAL,, &TOTAL&

MARCO
/: BOX XPOS 0 CM WIDTH '16.57' CM HEIGHT '12.47' CM FRAME 10 TW.

MY_ADD
AD IDES Holding AG
AD Neurottstrasse 16
AD Waldorf, 69190
AD Germany

PAGO
* FORMA DE PAGO

Programa de control

REPORT zform .

* VARIABLES

DATA BEGIN OF itcpo.
 INCLUDE STRUCTURE itcpo. "SAPscript Salida interfase
DATA END OF itcpo.

DATA: c_nif(10) TYPE c VALUE '12345678-Z',
 c_mensaje(74) TYPE c,
 n_reserva(5) TYPE n VALUE '12345',

```
total(10),  
subtotal(10),  
iva(10).
```

```
c_mensaje =  
'¡Nueva línea de productos !'.
```

```
DATA: BEGIN OF cabecera,  
    numero_factura(10) TYPE c VALUE '0025698094',  
    idcliente(10) TYPE c VALUE '5556981254',  
    descripcion(50) TYPE c VALUE 'Mario Lopez Alvarez',  
    fecha_pedido LIKE sy-datum VALUE '20030210',  
    fecha_factura LIKE sy-datum VALUE '20030214',  
END OF cabecera.
```

```
DATA: BEGIN OF i_posicion OCCURS 0,  
    posicion(3) TYPE c,  
    concepto(50) TYPE c,  
    cantidad TYPE i,  
    precio(5) TYPE c,  
END OF i_posicion.
```

* [Variables para comunicarnos con el formulario](#)

```
DATA: calle(40),  
    ciudad(20),  
    pais(20).
```

* [Configuración de la impresora](#)

```
PERFORM configurar_impresora.
```

* [Abrimos el formulario](#)

```
CALL FUNCTION 'OPEN_FORM'  
    EXPORTING  
        form = 'ZPRUEBA'  
    *   LANGUAGE = P_IDIOMA  
        options = itcpo  
        device = 'PRINTER'  
    *   DIALOG = 'X'  
        dialog = space " Sin diálogo  
    EXCEPTIONS  
        OTHERS = 1.
```

```
IF sy-subrc NE 0.  
    WRITE /error al abrir formulario'.  
    STOP.  
ENDIF.
```

* Carga el contenido de las variables
 PERFORM llenar_variables.

* Muestra el cuerpo del formulario
 PERFORM mostrar_cuerpo_form.

```
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    window = 'MARCO'
  EXCEPTIONS
    OTHERS = 1.
```

* Imprimimos la ventana dirección
 calle = 'C/ Modesto Lafuente 23'.
 ciudad = '28003, Madrid'.
 pais = 'SPAIN'.

```
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    window = 'ADDRESS'
  EXCEPTIONS
    OTHERS = 1.
```

```
IF sy-subrc NE 0.
  WRITE /error al escribir en ventana DIRECCION'.
  STOP.
ENDIF.
```

```
CALL FUNCTION 'CLOSE_FORM'.
*&-----*
*& Form CONFIGURAR_IMPRESORA
*&-----*
*   Configura los atributos de la impresora
*-----*
* --> p1
* <-- p2
*-----*
```

```
FORM configurar_impresora.
  itcpo-tdpageslct = space.      "Todas las páginas
  itcpo-tdnewid   = 'X'.        "Crea nuevo spool
  itcpo-tdcopies  = 1.          "1 copia
  itcpo-tddest    = 'LP01'.     "Nombre de la impresora
  itcpo-tdpreview = 'X'.        " Visualización previa
  itcpo-tdcover   = space.      "No portada
  itcpo-tdimmed   = 'X'.        "Imprime inmediatamente
  itcpo-tddelete  = 'X'.        "Borra después de imprimir
```

```
itcpcv-tdcovtitle = 'Ejemplo Formularios'.  
itcpcv-tdtitle    = 'Ejemplo Formularios'.
```

```
ENDFORM.                " CONFIGURAR_IMPRESORA  
*&-----*  
*&   Form LLENAR_VARIABLES  
*&-----*  
*   Da valores a las variables que se mostrarán en el formulario  
*-----*  
* --> p1  
* <-- p2  
*-----*  
FORM llenar_variables.  
  
* Llena la tabla interna de posiciones  
  i_posicion-posicion = '1'.  
  i_posicion-concepto = 'Papel fotocopidora (2500 hojas)'.  
  i_posicion-cantidad = 5.  
  i_posicion-precio = ' 8.50'.  
  APPEND i_posicion.  
  
  i_posicion-posicion = '2'.  
  i_posicion-concepto = 'Boligrafo tinta azul'.  
  i_posicion-cantidad = 100.  
  i_posicion-precio = '50.00'.  
  APPEND i_posicion.  
  
  i_posicion-posicion = '3'.  
  i_posicion-concepto = 'Portaminas 0.5'.  
  i_posicion-cantidad = 50.  
  i_posicion-precio = '99.35'.  
  APPEND i_posicion.  
  
  i_posicion-posicion = '4'.  
  i_posicion-concepto = 'Caja grapas (500)'.  
  i_posicion-cantidad = 25.  
  i_posicion-precio = '12.50'.  
  APPEND i_posicion.  
  
  i_posicion-posicion = '5'.  
  i_posicion-concepto = 'Hojas transparencias'.  
  i_posicion-cantidad = 15.  
  i_posicion-precio = '90.50'.  
  APPEND i_posicion.  
  
* TOTALES  
  total = '302.59 Euros'.
```

```
iva = ' 41.74'.  
subtotal = '250.85 Euros'.
```

```
ENDFORM.          " LLENAR_POSICIONES  
*&-----*  
*&   Form mostrar_cuerpo_form  
*&-----*  
*   text  
*-----*  
* --> p1   text  
* <-- p2   text  
*-----*
```

```
FORM mostrar_cuerpo_form .
```

```
* Imprimos la ventana principal  
CALL FUNCTION 'WRITE_FORM'  
  EXPORTING  
    window = 'MAIN'  
    element = 'ITEM_HEADER'  
  EXCEPTIONS  
    OTHERS = 1.
```

```
* Muestra las posiciones  
LOOP AT i_posicion.  
  CALL FUNCTION 'WRITE_FORM'  
    EXPORTING  
      window = 'MAIN'  
      element = 'DATOS'  
  EXCEPTIONS  
    OTHERS = 1.
```

```
ENDLOOP.
```

```
* Muestra los totales  
CALL FUNCTION 'WRITE_FORM'  
  EXPORTING  
    window = 'MAIN'  
    element = 'TOTAL'  
  EXCEPTIONS  
    OTHERS = 1.
```

```
* Imprime la zona de publicidad  
CALL FUNCTION 'WRITE_FORM'  
  EXPORTING  
    window = 'MAIN'  
    element = 'PUBLICIDAD'  
  EXCEPTIONS  
    OTHERS = 1.
```

ENDFORM. " mostrar_cuerpo_form

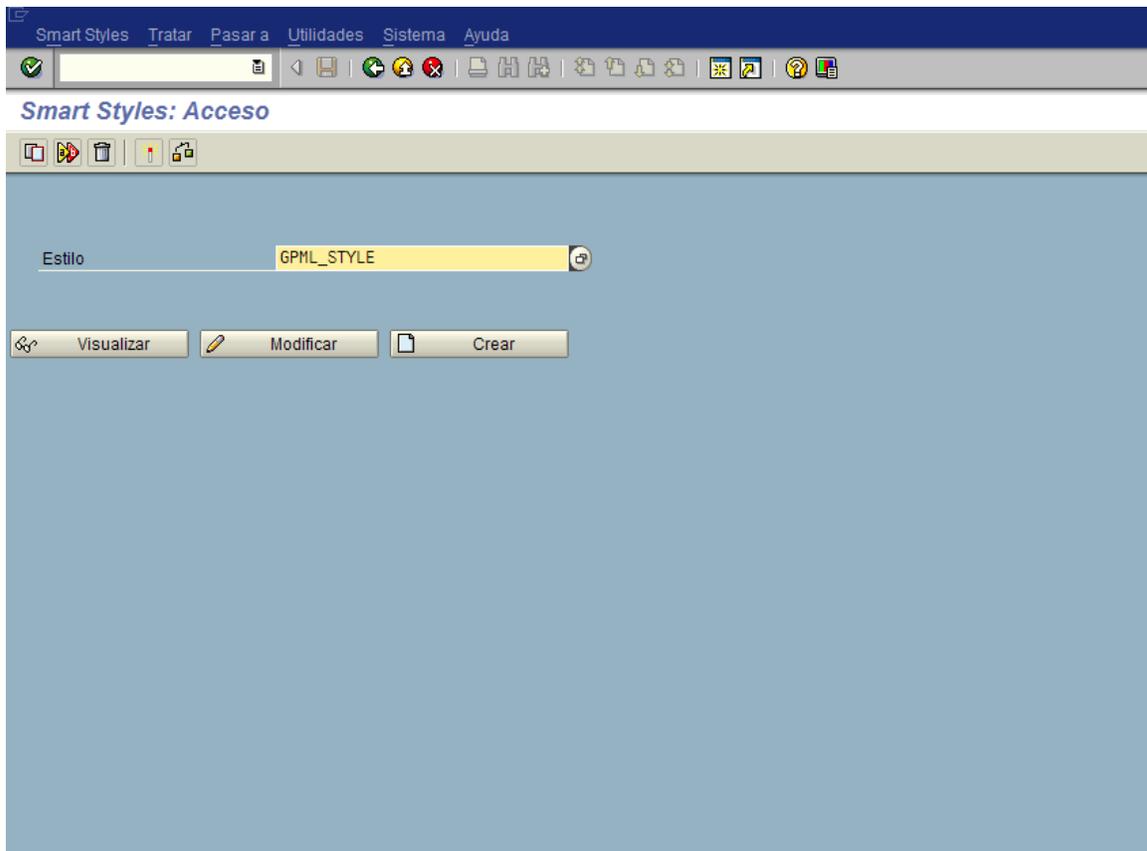
9.7. Smartforms

Se trata de una herramienta utilizada para la impresión y envío de informes e información tabulada y formateada a través de fax o correo electrónico.

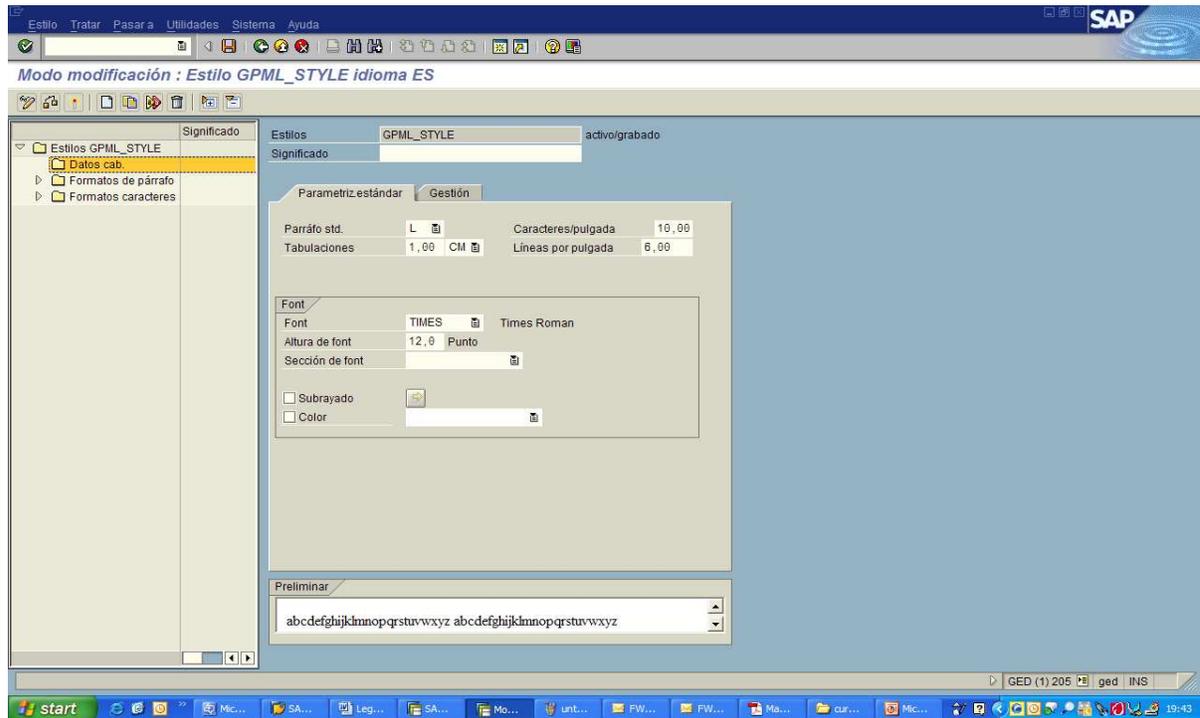
Para el tratamiento de los formularios creados con SMARTFORMS se utilizan dos transacciones:

- Una transacción para definir el estilo del formulario llamada SMARTSTYLES en la cual se definen los tipos de párrafos, tipos de caracteres, las fuentes que se van a usar, el tamaño que tendrán, las tabulaciones, ...

La pantalla que aparece al invocar a la transacción SMARTSTYLES es la siguiente:

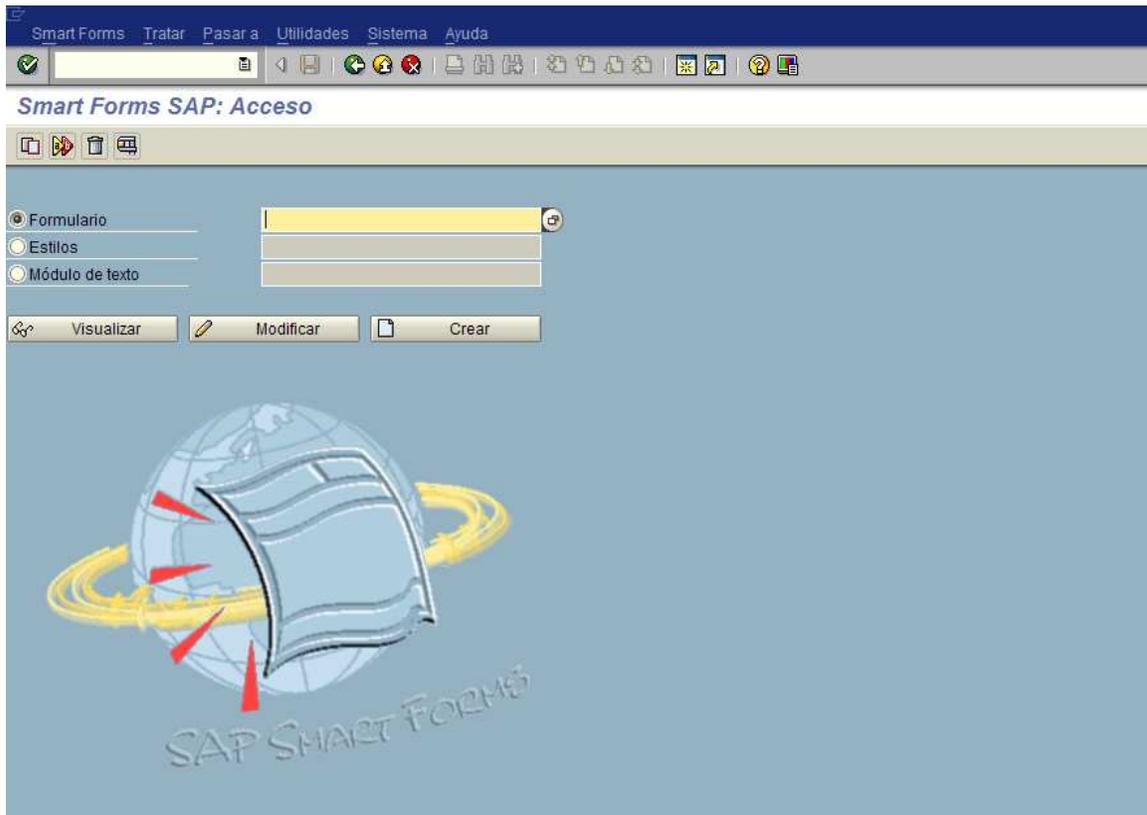


A continuación se muestra la pantalla de modificación de un estilo elegido en la transacción SMARTSTYLES.

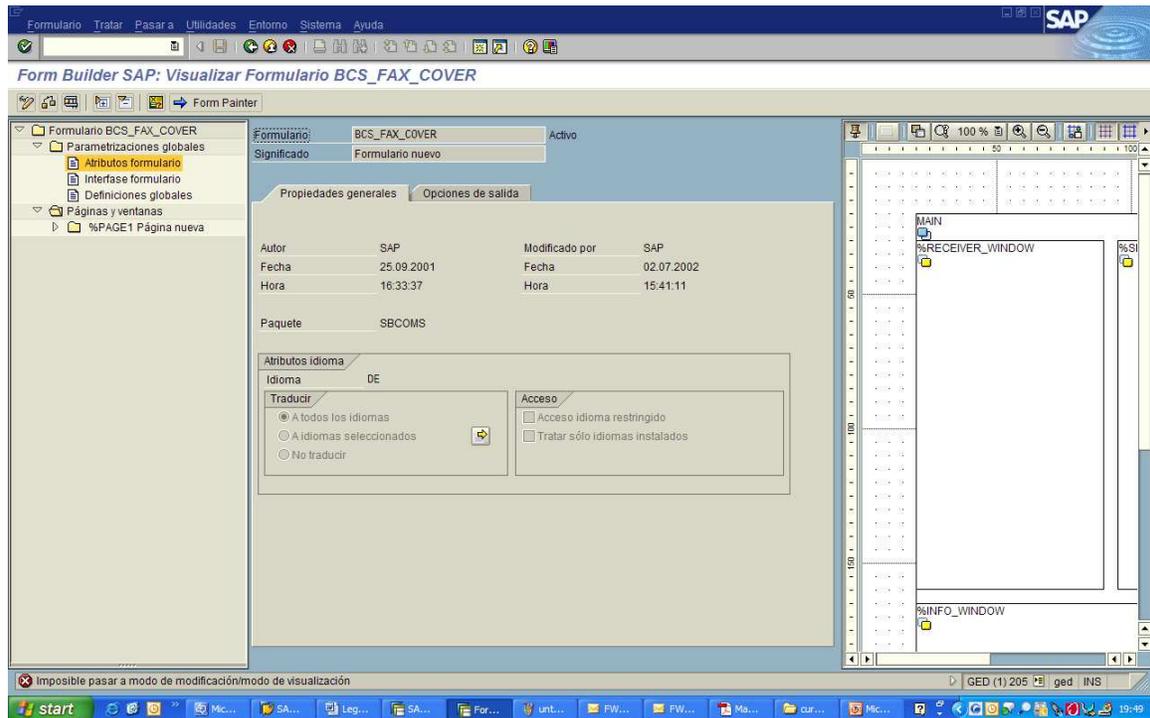


- La transacción para crear el formulario es la SMARTFORMS en la cual se definen las ventanas y su contenido. El estilo utilizado por el formulario es el creado con la transacción SMARTSTYLES.

La pantalla que aparece al invocar a la transacción se muestra a continuación. En ella se introduce el formulario que se desea editar, crear o visualizar.



A continuación se muestra una pantalla con la visualización de un formulario tomado como ejemplo:



10. Programación orientada a objetos

10.1 Introducción

La programación orientada a objetos es un método de desarrollo de software basado en el comportamiento real de los objetos en el mundo real. Se pretende desarrollar componentes de software que se comporten como los objetos reales a los que representan.

La orientación a objetos es una técnica usada en muchos lenguajes de programación los cuales comparten una terminología usada universalmente.

En esta sección se realizará una primera visión de conjunto de estos términos para en secciones posteriores adentrarse en la implementación de estos conceptos en el lenguaje ABAP.

10.2 Clases y objetos

Objeto

Un objeto es únicamente una porción de código fuente que contiene datos y proporciona servicios. Los datos constituyen los *atributos* del objeto. Los servicios que

proporciona el objeto se conocen como *métodos* y se asemejan en su funcionamiento a las funciones. Normalmente los métodos operan con los datos *privados* del objeto, esto es, con datos que son sólo *visibles* para los métodos del objeto. De esta manera, los atributos de un objeto no pueden ser cambiados directamente por el usuario del objeto, sólo pueden ser cambiados por los métodos de ese objeto. Así se garantiza la consistencia interna del objeto.

Uno de sus principales objetivos es el suministrar al desarrollador de software una forma de trabajo en la cual poder examinar un problema real y poder proporcionar una solución individualizada al problema. En el entorno de los negocios podrían ser objetos las entidades *Cliente*, *Factura*, etcetera.

¿Cómo crear objetos?

Antes de crear un objeto de una clase es necesario declarar una variable referenciada con la referencia a la clase. Una vez que se ha declarado la referencia <obj> a la clase <class>, se puede crear el objeto usando la sentencia **CREATE OBJECT <cref>**. Esta sentencia crea una instancia de la clase <class>, y la variable referenciada <cref> contiene la referencia al objeto.

Acceder a los componentes de un objeto

Los programas sólo pueden acceder a los componentes de las instancias usando las referencias de las variables referenciadas. La sintaxis es la siguiente, siendo ref la variable referenciada:

- Para acceder al atributo attr: **ref->attr**.
- Para llamar al método meth: **CALL METHOD ref->meth**.

Para los componentes estáticos (independientes de instancia, sólo dependientes de clase) se puede usar tanto el nombre de la clase como la variable referenciada. También es posible acceder a los componentes estáticos de una clase antes de que un objeto de la clase haya sido creado. La sintaxis, siendo class la clase es la siguiente:

- Para acceder al atributo estático attr: **class->attr**.
- Para llamar al método estático meth: **CALL METHOD class->meth**.

Dentro de una clase se puede acceder también a los componentes individuales mediante la referencia a sí mismo ME:

- Para acceder al atributo attr en la propia clase: **me->attr**.
- Para llamar al método meth en la propia clase:
CALL METHOD me->meth.

Clases

Una clase es una entidad teórica que describe el comportamiento de un objeto. Desde un punto de vista meramente técnico, un objeto es una *instancia* en tiempo de ejecución de una clase. En principio se pueden crear cualquier número de objetos basados en una

única clase. Cada instancia de una clase (objeto) tiene su propia identidad y su propio conjunto de valores para sus atributos. Dentro de un programa un objeto es identificado por su referencia, la cual le proporciona un nombre que define inequívocamente al objeto y permite acceder a sus métodos y atributos.

Las clases son las plantillas de los objetos. A la inversa, podemos decir que el tipo de un objeto es el mismo que el de su clase. Una clase es la descripción abstracta de un objeto. También podemos decir que una clase es un conjunto de instrucciones que tienen como objetivo construir un objeto. Los atributos de los objetos están definidos por los componentes de la clase (atributos, métodos y eventos), que son los que describen y controlan el comportamiento de los objetos.

10.2.1 Clases locales y globales

Las clases en ABAP Objects se pueden declarar bien globalmente o bien localmente. Las clases globales se definen en el generador de clases (transacción SE24) en el ABAP Workbench. Estas clases son almacenadas en class pools en la librería de clases en el R/3 Repository. Todos los programas ABAP en un sistema R/3 pueden acceder a las clases globales. Las clases locales se definen en un programa ABAP.

Las clases locales y sus interfaces sólo pueden ser invocadas desde el programa en el que se han definido.

Cuando se usa una clase en un programa ABAP el sistema busca primero una clase local con el nombre especificado. Si no encuentra ninguna entonces busca una clase global. A parte de la cuestión de la visibilidad, no hay ninguna diferencia entre usar una clase global o una clase local.

Definición de una clase local

Una definición completa de una clase constará de una parte declarativa en la que se definen los componentes, y si es necesario una parte de implementación en la que se implementan estos componentes.

La parte declarativa de una clase está comprendida entre las sentencias:

CLASS <class> DEFINITION.

...

ENDCLASS.

La parte declarativa contiene la declaración de todos los componentes de la clase (atributos, métodos y eventos). Cuando se definen clases locales, la parte declarativa pertenece a los datos globales del programa, por tanto se habrá de situar al principio del programa.

Si se declaran métodos en la parte declarativa de una clase, se deberá escribir también su parte de implementación. Ésta es la que va incluida entre las siguientes sentencias:

CLASS <class> IMPLEMENTATION.

...

ENDCLASS.

La parte de implementación contiene la implementación de todos los métodos de la clase. Esta parte actúa como un bloque, esto quiere decir que cualquier sección de código que no forme parte del bloque no será accesible.

10.3 Métodos y atributos

Métodos

Los métodos son procedimientos internos de una clase que definen el comportamiento de un objeto. Los métodos pueden acceder a todos los atributos de una clase. Esto les permite cambiar el contenido de los atributos de un objeto. Los métodos poseen también una interface con parámetros que les permite recibir valores cuando son invocados y devolver valores después de la llamada. Los atributos privados de una clase sólo pueden ser cambiados por métodos de la misma clase.

La definición y la interface de un método son similares a las de los módulos de funciones. Un método se define en la parte declarativa de la clase y se implementa en la parte de implementación usando las sentencias:

METHOD <meth>.

...

ENDMETHOD.

Se pueden declarar tipos de datos locales y objetos en los métodos de la misma manera que en cualquier otro procedimiento ABAP (subrutinas y módulos de funciones). Los métodos se pueden llamar mediante la sentencia **CALL METHOD**.

– Métodos dependientes de instancia – Estos métodos se declaran usando la sentencia **METHODS**.

Pueden acceder a todos los atributos de una clase, y pueden desencadenar todos los eventos de una clase.

– Métodos estáticos o independientes de instancia – Estos métodos se declaran usando la sentencia **CLASS-METHODS**. Sólo pueden acceder a los atributos estáticos y desencadenar eventos estáticos.

– Métodos especiales – Además de los métodos normales que se pueden llamar con la sentencia **CALL METHOD**, hay dos métodos especiales llamados **CONSTRUCTOR** y **CLASS_CONSTRUCTOR** que son automáticamente llamados cuando se crea un objeto (**CONSTRUCTOR**) o cuando se accede por primera vez a los componentes de la clase (**CLASS_CONSTRUCTOR**).

Atributos

Los atributos son los campos de datos internos de una clase y pueden tener cualquier tipo de datos ABAP.

El estado de un objeto viene determinado por el contenido de sus atributos. Un tipo de atributos son las variables referenciadas. Estas variables permiten crear y acceder a los

objetos, de manera que si se definen en una clase permiten acceder a otros objetos desde dentro de la clase.

– Atributos dependientes de instancia. – El contenido de estos atributos es específico de cada objeto. Se declaran usando la sentencia **DATA**.

– Atributos estáticos – El contenido de los atributos estáticos define el estado de la clase y es válido para todas las instancias la clase. Los atributos estáticos existen sólo una vez para la clase. Se declaran usando la sentencia **CLASS-DATA**. Son accesibles desde todo el entorno de ejecución de la clase. Todos los objetos de una clase pueden acceder a sus atributos estáticos. Si se cambia un atributo estático en un objeto, el cambio es visible en todos los demás objetos de la clase.

Ejemplo de declaración e implementación de una clase

CLASS C_CONTADOR DEFINITION.

PUBLIC SECTION.

**METHODS: FIJAR_CONTADOR IMPORTING VALUE(FIJAR_VALOR) TYPE I,
INCREMENTAR_CONTADOR,
OBTENER_CONTADOR EXPORTING
VALUE(OBTENER_VALOR) TYPE I.**

PRIVATE SECTION.

DATA CONT TYPE I.

ENDCLASS.

CLASS C_CONTADOR IMPLEMENTATION.

METHOD FIJAR_CONTADOR.

CONT = FIJAR_VALOR.

ENDMETHOD.

METHOD INCREMENTAR_CONTADOR.

ADD 1 TO CONT.

ENDMETHOD.

METHOD OBTENER_CONTADOR.

OBTENER_VALOR = CONT.

ENDMETHOD.

ENDCLASS.

10.4 Herencia

La herencia permite crear una nueva clase a partir de una nueva existente heredando la nueva clase sus propiedades. Esto se realiza añadiendo la adición **INHERITING FROM** a la sentencia de definición de la clase:

CLASS <subclass> DEFINITION INHERITING FROM <superclass>.

La nueva clase **<subclass>** hereda todos los componentes de la clase ya existente **<superclase>**.

La nueva clase se conoce como la subclase de la clase de la que procede. La clase original se conoce como la superclase de la nueva clase. Si no se añade ninguna declaración a la subclase, esta contiene los mismos componentes que la superclase. De cualquier manera, sólo los componentes públicos y privados de la superclase son visibles en la subclase. Aunque los componentes privados de la superclase existen en la subclase, no son visibles.

Se pueden declarar componentes privados en una subclase que tengan los mismos nombres que componentes privados de la superclase. Cada clase trabaja con sus propios componentes privados. Los métodos que una subclase hereda de una superclase usan los atributos privados de la superclase y no ningún componente privado de la subclase con el mismo nombre.

El nodo raíz de todos los árboles de herencia en ABAP Objects es la clase predefinida vacía **OBJECT**. Esta es la más general de todas las clases posibles ya que no contiene ni atributos ni métodos. Cuando se define una nueva clase no se tiene que especificar explícitamente esta clase como superclase, esta relación está definida implícitamente.

Redefinición de métodos

Todas las subclases contienen los componentes de todas las clases existentes entre ellas mismas y el nodo raíz del árbol de herencia. La visibilidad de un componente no puede ser cambiada nunca. En cambio se puede usar la adición **REDEFINITION** en la sentencia **METHODS** para redefinir un método público o protegido dependiente de instancia en una subclase y hacer que realice una función más especializada.

Cuando se redefine un método no se puede cambiar su interface, el método mantiene el mismo nombre y la misma interface de parámetros, pero tiene una nueva implementación. La declaración y la implementación de un método en una superclase no se ven afectadas cuando se redefine un método en una subclase. La implementación de la redefinición en la subclase 'oculta' la implementación original en la superclase.

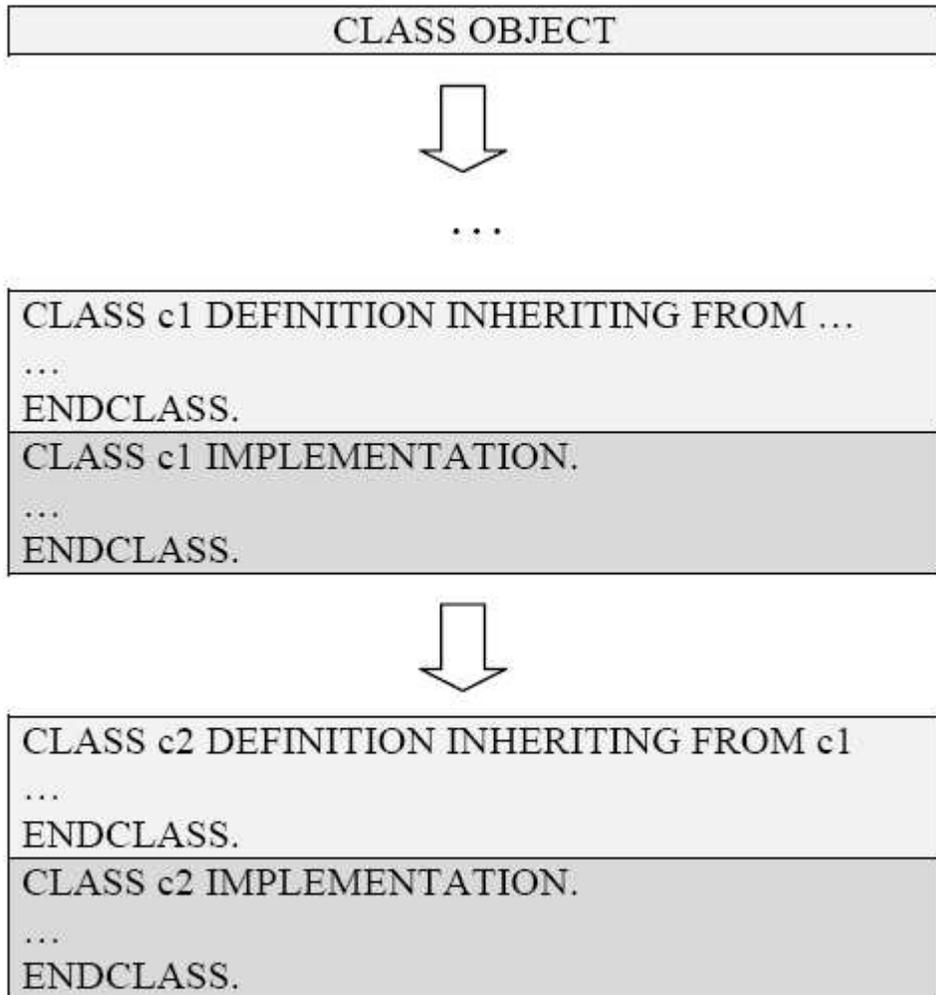
Clases y métodos abstractos y finales

Las adiciones **ABSTRACT** y **FINAL** en las sentencias **METHODS** y **CLASS** permiten definir métodos o clases abstractos y finales.

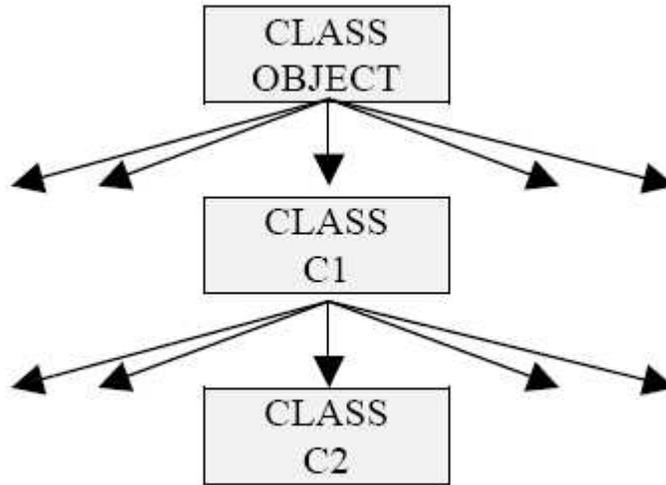
Un método abstracto se define en una clase abstracta y no puede ser implementado en esa clase, tiene que ser implementado en una subclase de la clase. Las clases abstractas no pueden ser instanciadas.

Un método final no puede ser redefinido en una subclase. Las clases finales no pueden tener subclases, son las que finalizan el árbol de herencia.

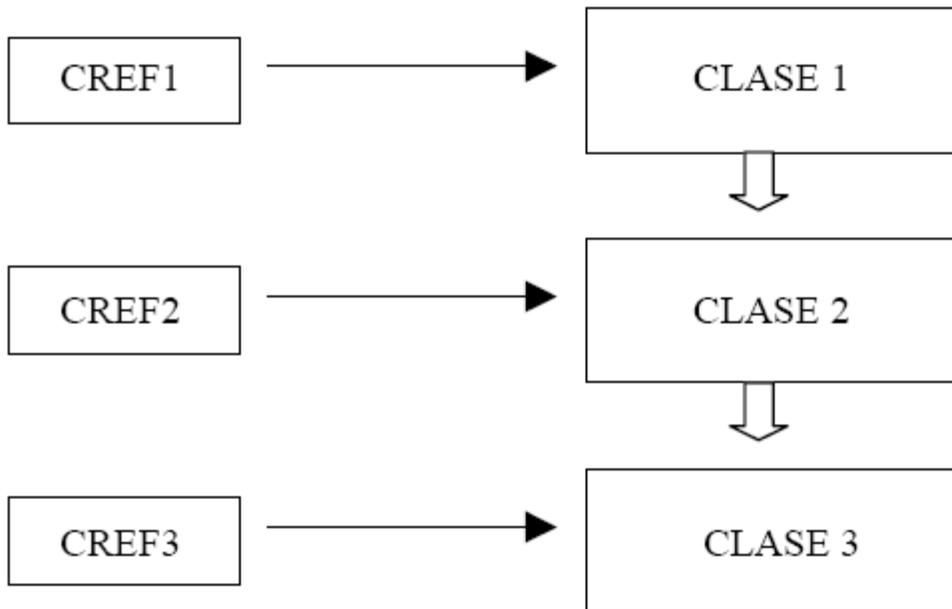
Herencia: Visión de conjunto



La subclase c2 deriva de la superclase c1. En lo más alto del árbol de herencia está la clase **OBJECT**.



La herencia simple consiste en que cada clase sólo deriva directamente de una superclase, pero puede tener varias subclases directas. La clase vacía OBJECT es el nodo raíz de cada árbol de herencia en ABAP Objects



Este gráfico muestra cómo variables referenciadas con referencia a una superclase pueden apuntar a objetos de sus subclases. Tenemos una instancia de la clase 3. Las variables referenciadas a clases cref1, cref2 y cref3 tienen el tipo de las clases 1, 2 y 3 respectivamente. Las tres variables referenciadas pueden apuntar a la clase 3, pero la variable cref1 sólo puede acceder a los componentes públicos de la clase 1, cref2

puede acceder a los componentes públicos de las clases 1 y 2, y cref3 puede acceder a los componentes públicos de todas las clases.

10.5 Polimorfismo

El polimorfismo quiere decir que métodos que se llaman exactamente igual pueden comportarse de manera distinta en clases diferentes. La orientación a objetos tiene unas estructuras llamadas interfaces que permiten acceder a métodos con el mismo nombre en diferentes clases.

Dentro de cada clase particular se puede redefinir el método obteniendo distintos métodos con el mismo nombre. Así es que un método no se define exactamente con su nombre, si no con su nombre y el nombre de la clase a la que pertenece.

Si se redefine un método dependiente de instancia en una o más subclases, se puede usar una única variable referenciada para llamar a las diferentes implementaciones de los métodos, dependiendo de la posición en el árbol de herencia del objeto referenciado. Este concepto de que diferentes clases pueden tener la misma interface y por lo tanto se puede acceder a ellas usando variables referenciadas con un único tipo se llama polimorfismo

10.6 Tablas internas de objetos

10.7 Ejemplos

Ejemplo: Cómo crear y usar una clase.

En este ejemplo veremos cómo crear y usar una instancia de la clase c_counter

DATA cref1 TYPE REF TO c_contador.

Creamos una variable **cref1** que es referenciada a la clase **c_contador**. Esta variable puede contener referencias a todas las instancias de la clase **c_contador**. La clase **c_contador** debe ser conocida para el programa en el momento en que la sentencia data tiene lugar. Por tanto la clase **c_contador** debe estar o bien declarada localmente antes de la sentencia data o bien globalmente con el constructor de clases.

Después de esta sentencia el contenido de **cref1** es **initial**, o sea la referencia no apunta a ninguna instancia.

DATA cref1 TYPE REF TO c_contador.

CREATE OBJECT cref1.

La sentencia **CREATE OBJECT** crea un objeto (instancia) de la clase **c_contador**. La referencia en la variable referenciada **cref1** apunta a este objeto.

La instancia de la clase **c_contador** se llama **c_contador<1>** debido a que así es como se visualizan los contenidos de la variable de objeto en el debugger después de que la sentencia **CREATE OBJECT** haya sido ejecutada. Este nombre es sólo usado internamente por el programa y no aparece nunca en el propio programa ABAP.

```
DATA cref1 TYPE REF TO c_contador.  
DATA numero TYPE i VALUE 5.  
CREATE OBJECT cref1.  
CALL METHOD cref1->fijar_contador  
EXPORTING fijar_valor = numero.  
DO 3 TIMES.  
CALL METHOD cref1->incrementar_contador.  
ENDDO.  
CALL METHOD cref1->obtener_contador  
IMPORTING obtener_valor = numero.
```

El programa ABAP puede acceder a los componentes públicos de los objetos usando la variable referenciada **cref1**, lo cual en este caso se corresponde a llamar a los métodos públicos de la clase **c_contador**. Después que el programa haya sido ejecutado la variable **numero** y el atributo privado del objeto **cont** tienen ambos el valor 8. Podemos también manejar varias instancias de la misma clase.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.
```

Así creamos tres variables referenciadas a la clase **c_contador**. Todas ellas contienen el valor **initial**.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
CREATE OBJECT cref1, cref2, cref3.
```

El sistema crea tres objetos de la clase a partir de las tres variables referenciadas a la clase. Las referencias en las tres variables apuntan a cada uno de los objetos. Internamente las instancias se llaman **c_contador <1>**, **c_contador <2>**, y **c_contador <3>**. El número se asigna en el orden en que son creadas.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
DATA numero1 TYPE i VALUE 5.  
DATA numero2 TYPE i VALUE 0.  
DATA numero3 TYPE i VALUE 2.
```

```
CREATE OBJECT cref1, cref2, cref3.  
CALL METHOD: cref1->fijar_contador  
EXPORTING fijar_valor = numero1,  
cref2->fijar_contador  
EXPORTING fijar_valor = numero2,  
...  
CALL METHOD cref2->incrementar_contador.
```

```
...  
CALL METHOD cref1->incrementar_contador.
```

```
...  
CALL METHOD: cref1->obtener_contador  
IMPORTING obtener_valor = numero1,  
 cref2->obtener_contador  
IMPORTING obtener_valor = numero2,
```

```
...
```

El programa ABAP usa las variables referenciadas para acceder a los objetos, en este caso a los métodos públicos de la clase **c_contador**.
Cada objeto tiene su propia identidad y su propio estado, ya que el atributo privado dependiente de instancia **cont** tiene distintos valores en cada objeto. El programa administra varios contadores.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
CREATE OBJECT cref1, cref2.
```

Ahora declaramos tres variables referenciadas para la clase **c_contador** y se crean dos objetos para la clase. Las referencias en las variables referenciadas **cref1** y **cref2** apuntan a cada uno de los objetos.
La referencia **cref3** se mantiene **initial**.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
CREATE OBJECT cref1, cref2.  
MOVE cref2 TO cref3.
```

Después de la sentencia **MOVE**, **cref3** contiene la misma referencia que **cref2** y ambas referencias apuntan al objeto **c_contador<2>**. Un usuario puede usar cualquiera de ellas para acceder al objeto.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
CREATE OBJECT cref1, cref2.  
MOVE cref2 TO cref3.  
CLEAR cref2.
```

La sentencia **CLEAR** reinicializa la referencia de **cref2** al valor **initial**. En este momento la variable referenciada **cref2** contiene el mismo valor que inmediatamente después de su declaración y ya no apunta a ningún objeto.

```
DATA cref1 TYPE REF TO c_contador.  
DATA cref2 TYPE REF TO c_contador.  
DATA cref3 LIKE cref1.  
CREATE OBJECT cref1, cref2.  
MOVE cref2 TO cref3.  
CLEAR cref2.  
cref3 = cref1.
```

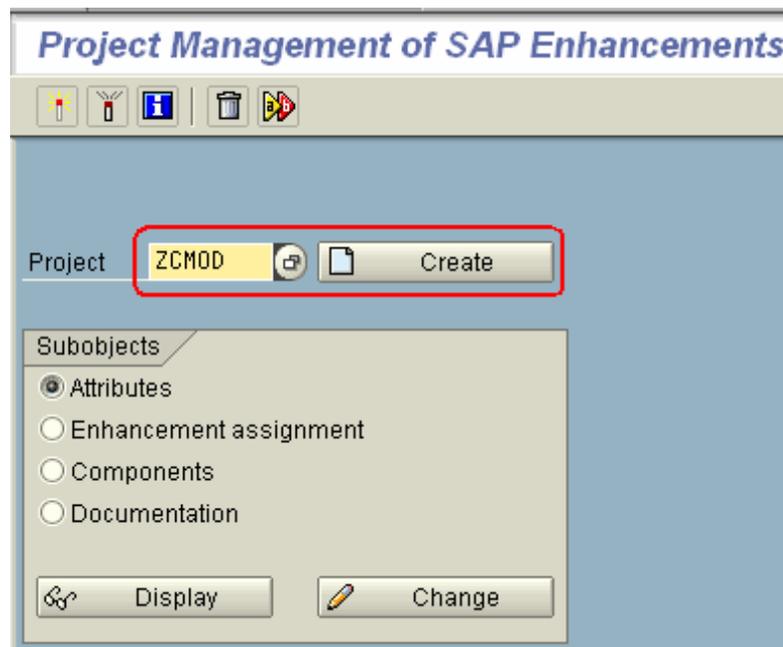
La referencia en **cref3** ahora apunta al objeto **c_contador<1>**. Ya no hay referencias apuntando al objeto **c_contador<2>** el cual es automáticamente borrado, con lo cual el nombre interno **c_contador<2>** está libre de nuevo.

11- AMPLIACIONES SAP

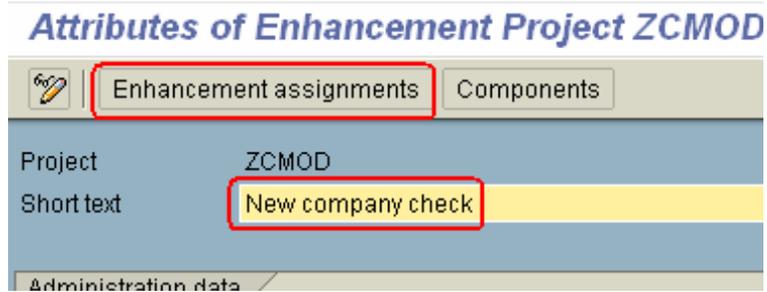
11.1CMOD y SMOD

Utilizamos las transacciones CMOD y SMOD para para gestionar las user exits. Antes de implementar una exit es necesario crear un proyecto a través de la transacción CMOD.

Si queremos por tanto crear una exit, vemos cómo creamos el proyecto en la **CMOD**. Lo veremos con capturas de pantallas.

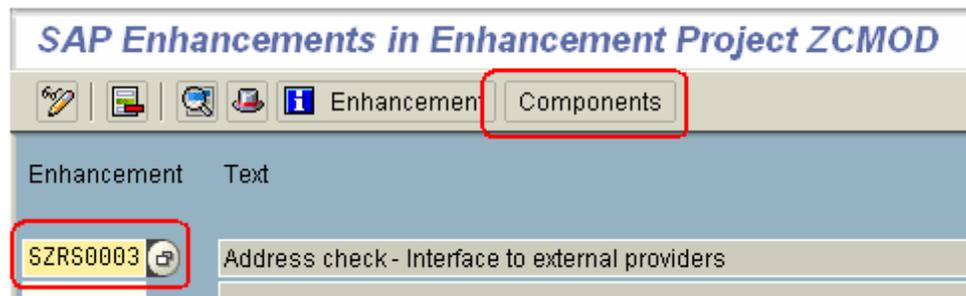


Especificamos un nombre de proyecto, en este caso el CMOD



Damos al proyecto una descripción y lo guardamos. Ahora necesitaremos añadir nuestra ampliación , para ello deberemos pulsar el botón “Asignación de ampliación” o “Enhancement assignments”.

Añadimos la ampliación “SZRS0003” y guardamos. Ahora presionamos el botón “Componentes ” para ver las posibles modificaciones de esta ampliación.



De la lista de componentes mostrada abajo, podemos ver que contamos con una única user exit. Además podemos apreciar que el el proyecto aún no está activado, esto nos lo indica el icono rojo, que cambiará a verde cuando activemos el proyecto.

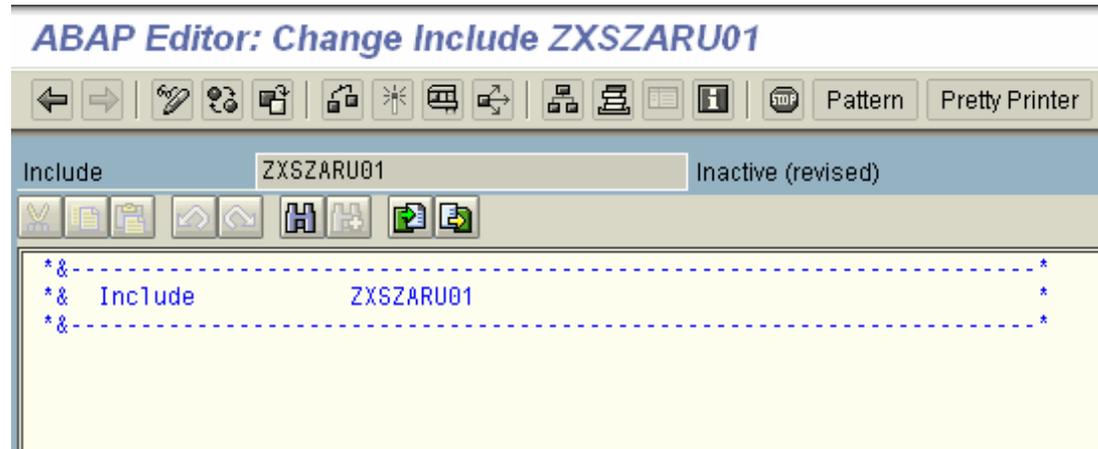
The screenshot shows the 'Change ZCMOD' interface. The 'Enhancement assignments' tab is selected. Below the tab, there is a table with the following data:

Project				ZCMOD New company check
Enhancement	Impl		Exp	SZRS0003 Address check - Interface to external providers
Function exit				EXIT_SAPLSZAR_001

Para implementar nuestra exit deberemos hacer doble clic sobre la exit. Esto nos lleva a la transacción de visualización funciones SE37, en la que podemos ver que hay una única línea dentro de la función, es un incluye

```
include zxszaru01 .
```

Este incluye comienza con "Z" (espacio reservado al usuario), lo que nos permite mantener nuestro código separado del de sap. Hemos doble click sobre en incluye lo que nos lleva al editor de programas ABAP (SE38).



Una vez aquí ya podremos implementar nuestra ampliación.

11.2 USER EXIT

Lugares predeterminados en el código estándar de SAP que permiten al usuario introducir funcionalidades adicionales que SAP puede no proporciona en dicho código estándar. Esto permite al programador adaptar SAP a los requerimientos específicos de su proyecto, sin tener que modificar el código de los programas estándar.

11.3 BADIS

Las **BADI's (Business Ad-ins)** son una herramienta de programación ABAP orientada a objetos que se utilizan en SAP para implementar validaciones y ampliaciones en el código estándar de SAP en versiones a partir de la 4.6c. Es decir, sirven para acomodar los requerimientos específicos de un cliente a las transacciones estándar de SAP.

El código generado por SAP en sus transacciones estándar (para realizar un pedido...) no se puede modificar (salvo para implementar un parche de SAP), ya que se perdería el soporte que este ofrece a su producto. Pero supongamos que cuando se termina de hacer un pedido de compra por medio de la transacción ME21N necesito guardar ciertos datos de este pedido en una tabla ZPEDIDOS que ha sido creada previamente.

Pues para esto están las ampliaciones (BADI's, user exits, field exits) que al fin y al cabo no son más que fragmentos de código que me permite SAP meter dentro de su código estándar para realizar ciertas operaciones a medida, en este caso meter la información que necesito en una tabla cuando se crea el pedido en concreto.

Básicamente cumplen la misma función que las **USER EXITS** y las **FIELD EXITS** amén de algunas diferencias que se exponen a continuación.

Diferencia entre BADI y las USER EXITS

- **BADI** se puede utilizar todas las veces que quieras, donde las users exits solo se pueden usar una sola vez. Ejemplo: Si se asigna una user-exit a un proyecto usando la transacción **CMOD**, entonces no se puede asignar a ningún otro proyecto. Es decir varios programadores pueden implementar la misma BADI independientemente.
- Las BADI's son mucho más moldeables a las necesidades del programador en cuanto a que se pueden definir los puntos de salida, junto con la lógica de programación que necesitemos. Es decir posee todas las propiedades de una programación orientada a objetos

12- RESTO DE INSTRUCCIONES ABAP



C:\Documents and Settings\rdemiguel\M\

13- TABLAS ESTANDAR SAP

TABLAS DE DATOS

A continuación se detalla el nombre de las tablas SAP más utilizadas, clasificadas por módulos

FI

AGKO Cuentas compensadas
AVIK Cabecera de aviso
AVIP Pos. de aviso
AVIR Subposición de aviso
AVIT Cabecera de aviso: texto de libre definición Maestros
BKPF Cabecera de documento para Contabil.
BSAD Cont.: índice sec. para deudores (partidas compensadas)
BSAK Cont.: índice sec. para acreedores (partidas compensadas)
BSAS Cont.: índice secundario para ctas. mayor (partidas compensadas)
BNKA Maestro de banco
BSBM Campos de valoración del documento
BSBW Campos de valoración del documento
BSEC Segm. de documento para datos CPD
BSED Segm. de documento - campos de efectos

BSEG Segm. de documento de Contabilidad
BSES Datos de control de documento
BSET Datos de control de segm de documento
BSID Cont.: índice secundario de deudores
BSIK Cont.: índice secundario de acreedores
BSIP Índice para verificación de documentos dobles de acreedores
BSIS Cont.: índice secundario ctas.mayor
BVOR Operaciones contables multisociedades
PAYR Fichero de medio de pago
PCEC Cheques prenumerados
PNBK Notific. Previa nuevos datos bancarios de regs.
REGUH Datos de pago del programa de pagos
REGUP Posic. tratadas del programa de pagos

SD

TVDC Plazos de entrega: Clases
VBAG Documento ventas: Fecha para repartos en planes de entrega
VBAK Documento de ventas: Datos de cabecera
VBAP Documento de ventas: Datos de posición
VBEH Historial repartos entrega
VBEP Documento de ventas: Datos de reparto
VBKD Documento de ventas: Datos comerciales
VBLB Doc. de ventas: Datos ord.-entrega
VBSN Estados de modificación p. pl.-entregas
VBSP Documento RV posición para ejecutar estructura de artículo
VBFA Flujo de documentos de ventas
VBPA Documento Comercial: Interlocutor
VEDA Datos contractuales
VLPKM Planes entrega para mat.de cliente
LIKP Doc.comercial: Entrega - Datos de cabecera
LIPS Doc.comercial: Entrega - Datos de posición
VBUP Documento comercial: Status de posición
VBUK Documento comercial: Status cabecera y datos gestión
VBRK Doc.comercial: Factura: Datos de cabecera
VBRP Doc. Comercial: Factura: Datos de posición
VTTK Cab. Transporte
VTTP Posición de transporte
VTTS Etapa de transporte

CO

COSP Objeto CO: Totales de costes – contabilizaciones externas
COBK Objeto CO: Cabecera de documento
COEP Objeto CO: Partida individual por períodos
CSKS Maestro de centros de costo
CSKB Clases de Costos
CKHS Cabecera – cálculo de costes unitario (control + sumas)
CKIS Posiciones del CCU/Vis. Detallada costes del producto
COKP Objeto CO: Cabecera del documento
COEJ Objeto CO: partida ind. por año
CSKA Clases de coste (datos dependientes del plan ctas)
CSKB Clases de coste (datos dependientes de la sociedad CO)

CA

AUSP Valores prop. de las características
CABS Resultado evaluación estad. de AUSP
KLAH Datos cabecera clase
KLAT Clases: Textos explicativos
KSML Características de clases

KSSK Tabla de asignación: Objeto a clase
CABN Característica
CAWN Valores de características
TCLA Categorías clase
TCLAO Varios objetos en una categoría de clase
TCLAT Textos categorías de clase
TCLC Status de clasificación
TCLD Datos característ. de norma
TCLG Grupos de clases
TCLO Campos clave de objetos
CDHDR Cabecera en el documento de modificación
CDPOS Posiciones en el documento de modificación
KONV Condiciones (Datos operación) -- (Precios de las facturas)
STXH SAPscript Cabecera fichero texto
STXL SAPscript Líneas del fichero de texto

MM

EKKO Cabecera de documento de compras
EKPO Posiciones de documento de compras
EBNA Solicitud de pedido
MKPF Cabecera de documentos de materiales
MSEG Posiciones de documento de materiales
MARA Maestro materiales general
MARC Materiales por centro
MARD Almacén por centro
ABEW Valoraciones de los materiales
EKKN Imputación en documentos de compra
MARM Unidad de medida
EINA Registro info de compras: Datos generales
RBKP Cabecera documento factura recibida

PP

AFFL Secuencia orden de trabajo
AFKO Datos cabecera orden para órdenes PCP
AFPO Posición de orden
AFVC Operación de orden
FAPW Índice de centro de producción / suministro para orden fabricación
CRCA Asignación de capacidad al puesto de trabajo
CRCO Asignación de puesto de trabajo a centro de coste
CRHD Cabecera del puesto de trabajo
CRHH Datos cabecera jerarquía
CRHS Jerarquía Estructura
CRID Recursos de producción – Entidades
CRTX Puesto de trabajo / Denominación breve del medio auxiliar fabric.
KAKT Denominación de la capacidad
KAKO Capacidad segmento de cabecera
MAPL Asignación de hojas de ruta para materiales
PLAS Hoja de ruta - Selección de posiciones
PLFH Hoja de ruta - Medios auxiliares de fabricación
PLFL Secuencias hoja de ruta
PLKO Hoja de ruta – Cabecera
PLKZ HRuta: cab.central
PLMZ Asignación de posiciones de lista materiales a operaciones
PLPO Hoja de ruta – operación
KAPA Capacidad - Valores del turno
KAPE Capacidad asignación unidad de base
KAZY Capacidad - Intervalo de oferta

PS

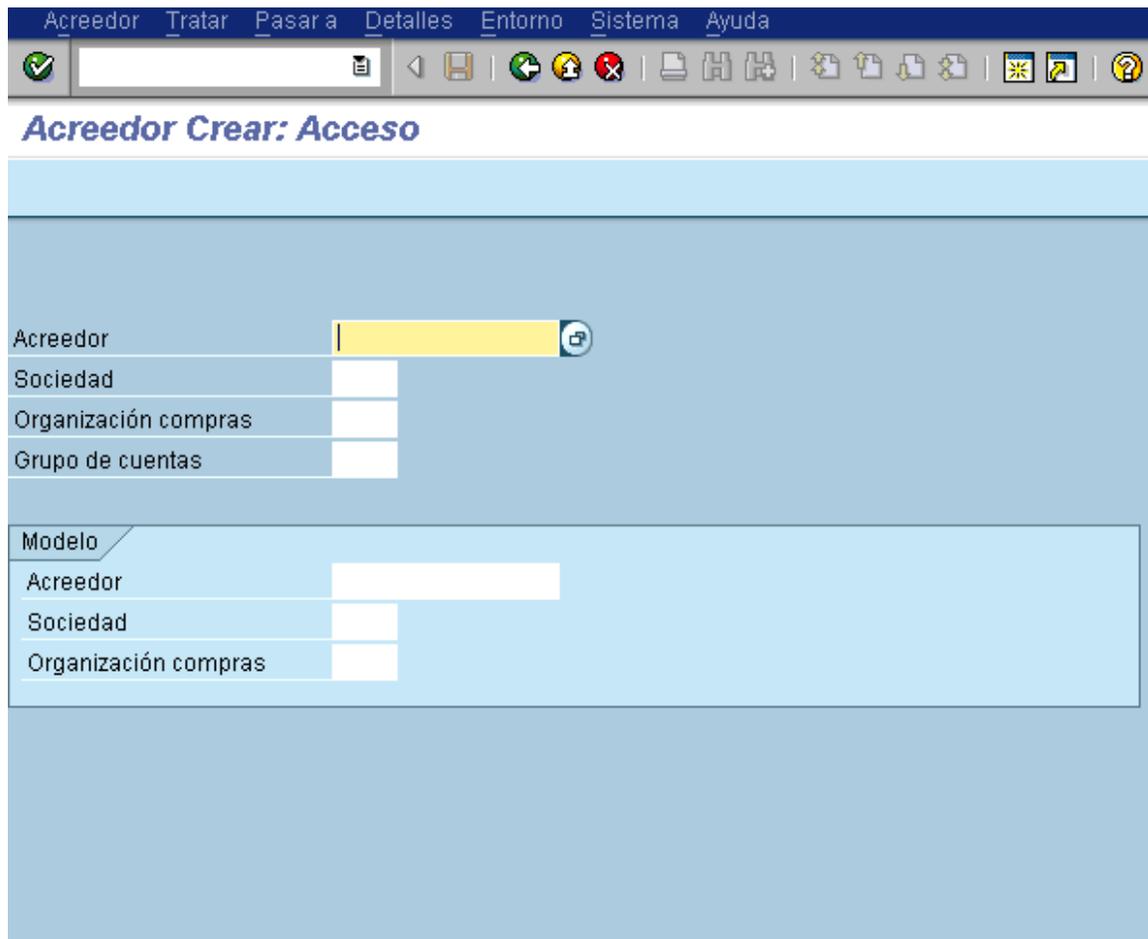
AFAB Grafo - relaciones de ordenación
AFRV Pool de notificaciones
NPTX Textos PS (grafo)
PLAB Plan - relaciones de asignación
PLTX Textos PMS (grafo estándar)
PRHI Plan de estr.de proyectos, arcos (indicador de jerarquías)
PRHS PEP estándar, arcos (puntero jerarquía)
PROJ Definición del proyecto
PROJS Definición de proyecto estándar
PRPS Elemento PEP (elem.de plan estruct.proyecto) datos maestros
PRPSS Elemento PEP estándar (Elemento PEP) – Datos maestros
PRTE Fechas de programación posición de proyecto
PRTP Subproyectos
PRTX Textos PMS (PEP)
PRTXS Textos PS (PEP estándar)
PSERB Datos de herencia: sistema de proyectos
LFINF Informaciones entrega sistema de proyectos
LFINFX Asign.informaciones entrega para objetos sistema proyectos
MLST Hito
MLTX Denominación de etapa
MSET Set etapas estándar
NHPROJ Historial de números - Definición de proyecto
NHPRPS Reorg. historial de números - Elementos PEP
NPTX Textos PS (grafo)

14- TRANSACCIONES ESTANDAR SAP

14.1 Obtención de la ayuda técnica (F1)

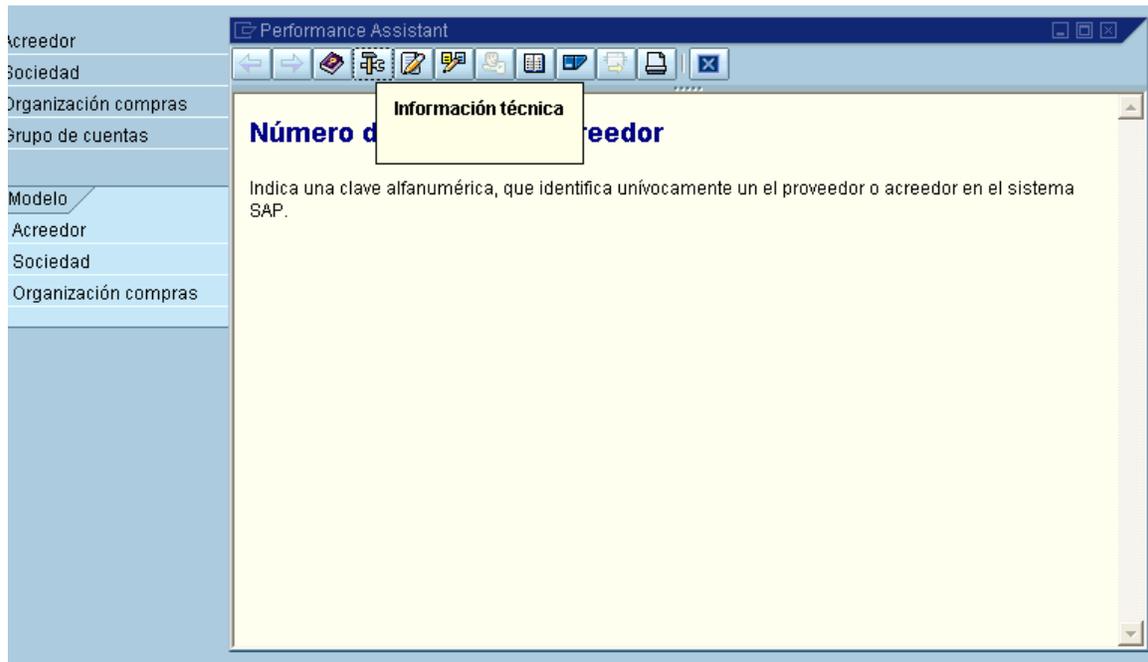
En este apartado explicaremos la manera de obtener los datos técnicos de cualquier campo que aparezca en una pantalla de SAP determinada.

Por ejemplo, entramos en la transacción XK01, utilizada para la creación de proveedores.

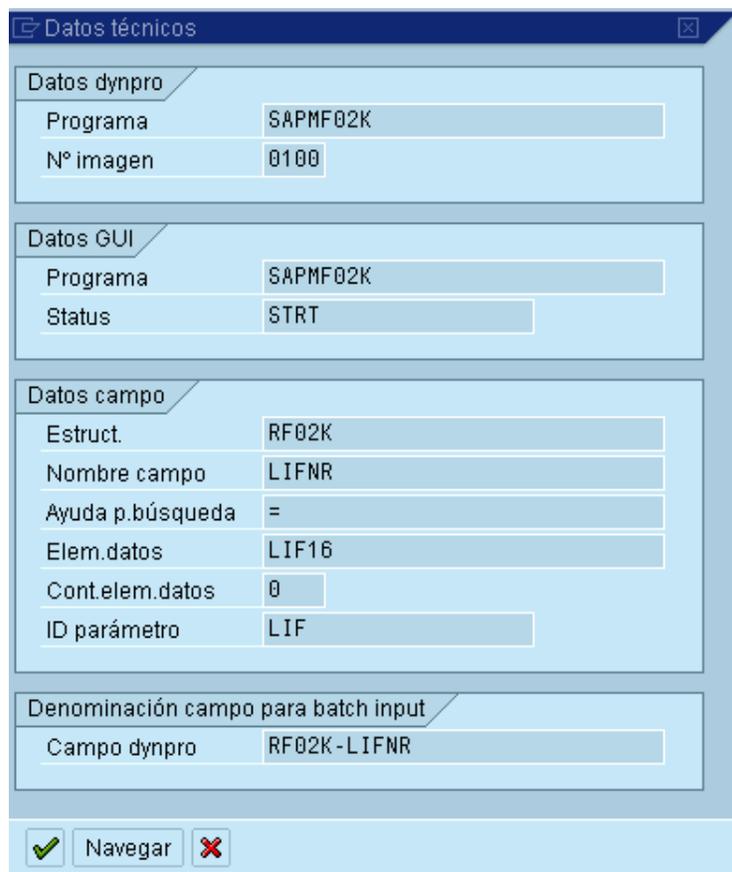


Imaginemos que queremos conocer los datos técnicos del campo Acreedor. Nos posicionáramos con el ratón en el campo y pulsáramos la tecla de función **F1**.

La siguiente pantalla que aparece es el Performace Assistant. Ahí podemos encontrar, entre otra información, los datos técnicos del campo, pulsando el botón que aparece señalado en la captura de pantalla que mostramos a continuación (Información técnica). Igualmente, aparecerá una breve descripción del campo que estamos explorando.



Al pulsar sobre este botón, la pantalla que se nos presenta es la siguiente:



Aquí encontramos datos técnicos referentes al campo y a la pantalla en la que está incluido, los analizamos:

Datos dynpro

Datos referentes a la pantalla.

- Programa: El nombre del programa estándar que invoca a la pantalla
- N° imagen: El número de pantalla.

Datos GUI

Datos referentes a la interfaz gráfica _

- Programa: El nombre del programa estándar que invoca a la pantalla
- Status: Estado.

Datos Campo

Datos referentes al campo que hemos seleccionado

- Estruct: La estructura que contiene el campo. Existen casos en los que aparece directamente la tabla a la que pertenece, en lugar de la estructura.
- Nombre del campo: Nombre técnico del campo
- Elemento de datos: Tipo de elemento de datos del campo en cuestión
- ID parámetro: ID al que está vinculado ese campo.

Denominación campo para batch input

El nombre del campo en cuestión en la tabla BDC-data generada tras hacer un batch input.

14.2 Trace del Sistema (ST05)

La transacción ST05 nos permite generar una traza del sistema. Es decir, visualiza todas las acciones que se dan lugar en el sistema, desde un momento determinado (cuando activamos la traza) hasta otro (cuando la desactivamos).

La captura de pantalla nos muestra la pantalla que se presenta cuando entramos a la transacción ST05. A esta transacción podremos acceder de dos maneras distintas:

Directamente → ST05

Por menú: Sistema → Utilidades → Traza de rendimiento

De las dos maneras el resultado es el mismo:

Performance Analysis

Select Trace

- SQL Trace
- Enqueue Trace
- RFC Trace
- Buffer Trace

Select Trace Function

- Activate Trace
- Activate Trace with Filter
- Deactivate Trace
- Display Trace
- Enter SQL Statement

Trace Status

Pulsaremos el botón “ACTIVATE TRACE”, para activar la traza.

A continuación llevaremos a cabo la acción que queremos analizar mediante la traza. Entrar a una transacción, realizar una grabación....

Una vez realizadas las acciones que queremos analizar, pulsaremos el botón “DEACTIVATE TRACE”.

Para visualizar la traza, deberemos pulsar el botón “DISPLAY TRACE”, y aparecerá una pantalla como la que se muestra a continuación:

Trace List

Transaction XK01		Work process no 0	Proc.type	Client 205	User DESARROLL06
Duration	Objeto		RC	Statement	
6.495	ADCNTYQU	FETCH	0	1403	
220	ATAB	PREPARE	0	0	SELECT WHERE "TABNAME" = :A0 AND "VARKEY" LIKE :A1 ORDER BY "TABNAME" , "V
5	ATAB	OPEN	0	0	SELECT WHERE "TABNAME" = 'T077Y' AND "VARKEY" LIKE '205%' ORDER BY "TABNAM
39.950	ATAB	FETCH	62	0	
5.577	ATAB	FETCH	49	1403	
225	NRIV	PREPARE	0	0	SELECT WHERE "CLIENT" = :A0 AND "OBJECT" = :A1 AND "SUBOBJECT" = :A2 AND "
5	NRIV	OPEN	0	0	SELECT WHERE "CLIENT" = '205' AND "OBJECT" = 'KREDITOR' AND "SUBOBJECT" =
26.629	NRIV	FETCH	1	0	
14	NRIV	REOPEN	0	0	SELECT WHERE "CLIENT" = '205' AND "OBJECT" = 'KREDITOR' AND "SUBOBJECT" =
1.750	NRIV	FETCH	1	0	
939	T078K	PREPARE	0	0	SELECT WHERE "MANDT" = :A0 ORDER BY "MANDT" , "TCODE"
5	T078K	OPEN	0	0	SELECT WHERE "MANDT" = '205' ORDER BY "MANDT" , "TCODE"
14.546	T078K	FETCH	9	1403	
203	T079K	PREPARE	0	0	SELECT WHERE "MANDT" = :A0 ORDER BY "MANDT" , "BUKRS"
4	T079K	OPEN	0	0	SELECT WHERE "MANDT" = '205' ORDER BY "MANDT" , "BUKRS"
56.434	T079K	FETCH	1	1403	
212	T079M	PREPARE	0	0	SELECT WHERE "MANDT" = :A0 ORDER BY "MANDT" , "EKORG"
4	T079M	OPEN	0	0	SELECT WHERE "MANDT" = '205' ORDER BY "MANDT" , "EKORG"
9.469	T079M	FETCH	2	1403	
386	TSAD9V	PREPARE	0	0	SELECT WHERE "CLIENT" = :A0 ORDER BY "CLIENT" , "NATION"
5	TSAD9V	OPEN	0	0	SELECT WHERE "CLIENT" = '205' ORDER BY "CLIENT" , "NATION"
3.589	TSAD9V	FETCH	0	1403	
228	ATAB	PREPARE	0	0	SELECT WHERE "TABNAME" = :A0 AND "VARKEY" BETWEEN :A1 AND :A2 ORDER BY "TA
5	ATAB	OPEN	0	0	SELECT WHERE "TABNAME" = 'T100C' AND "VARKEY" BETWEEN '205F2' AND '205F2
9.409	ATAB	FETCH	1	1403	
193	ATAB	PREPARE	0	0	SELECT WHERE "TABNAME" = :A0 ORDER BY "TABNAME" , "VARKEY"
5	ATAB	OPEN	0	0	SELECT WHERE "TABNAME" = 'TFMC' ORDER BY "TABNAME" , "VARKEY"
4.981	ATAB	FETCH	7	1403	

En ella podemos ver todos los accesos que se han llevado a cabo en el sistema mientras hemos creado a un proveedor, a través de la transacción XK01.

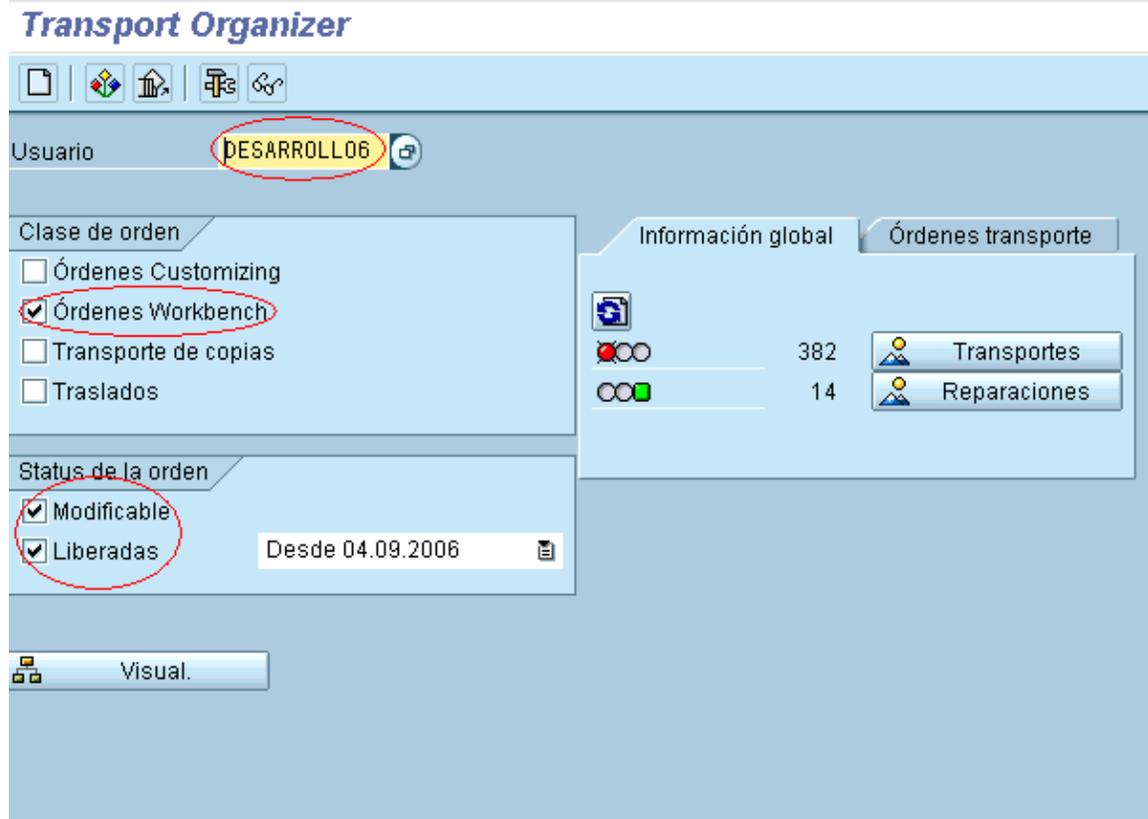
14.3 Transporte (se10 y STMS)

Las transacciones SE10 y STMS se utilizan por ejemplo, cuando se quiere ejecutar el programa que hayamos creado en un mandante distinto al mandante en el que lo estamos desarrollando.

La transacción SE10 permite liberar las órdenes de transporte en la que se encuentre el programa que vamos a probar. No se podrá transportar ninguna orden que no haya sido liberada anteriormente.

La transacción STMS es la que se encarga del transporte de las órdenes desde el mandante fuente al mandante destino.

Las capturas de pantalla siguientes nos muestran su funcionamiento:



A la hora de liberar las órdenes es importante que seleccionemos el tipo de orden que vamos a liberar, si es de workbench o de customizing, y si está o no liberada. Es decir, si vamos a liberarla, debemos seleccionar la opción “Modificable” y deseleccionar la opción “Liberadas”, y aparecerán todas las órdenes que no estén liberadas. A continuación pulsamos el botón “Visual.”

Transport Organizer: Órdenes

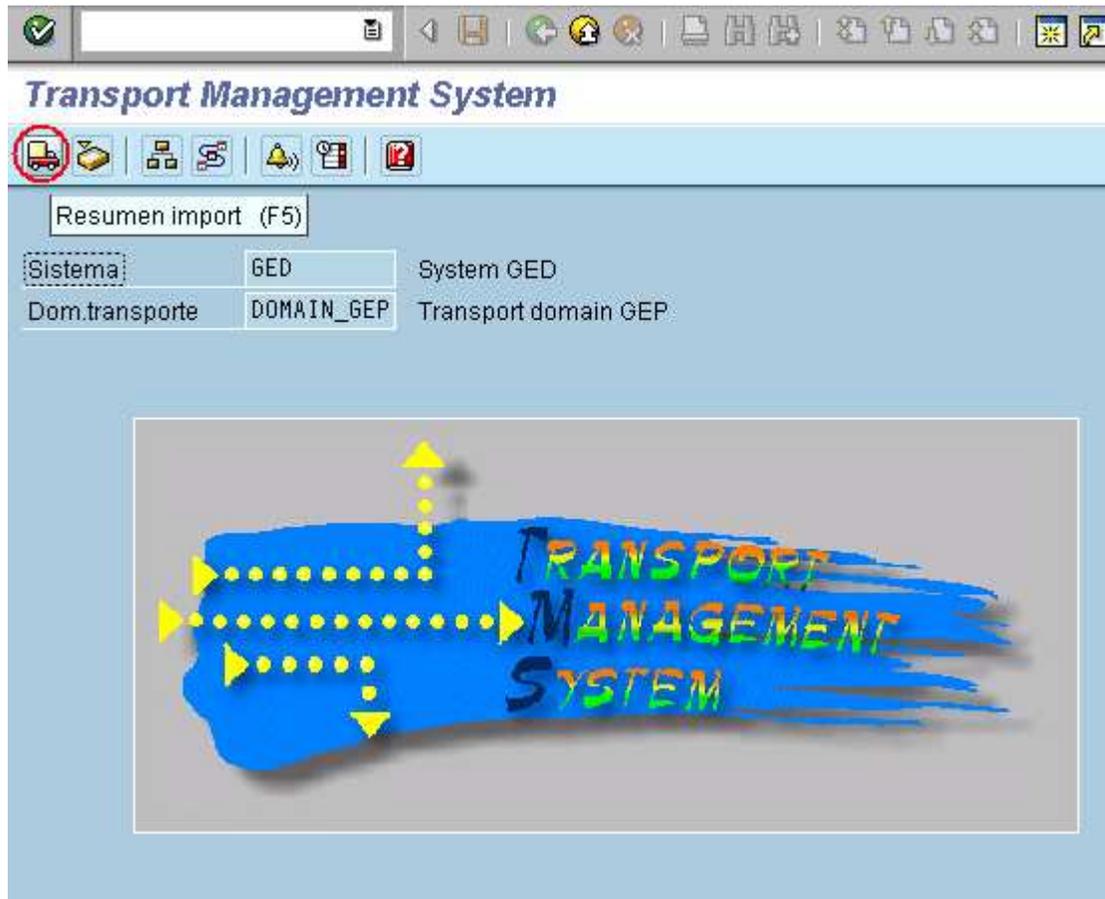
Órdenes Workbench con participación de DESARROLLO6 (DESARROLLO6)

Order ID	System	Description
205	Copia del 900	
-> GEI	Sistema GEI	
Modificable		
GEDK925256	DESARROLLO6	RSIIN: Modificación masiva de roles #27
GEDK926252	DESARROLLO6	FASE II: Grupo de Funcion Carga de Clientes y Proveedores
GEDK926084	DESARROLLO6	RSII: Borrado de roles #1
GEDK926044	DESARROLLO6	RSIIB: Transacción del programa de roles
GEDK925491	DESARROLLO6	MatchCode para tabla Z_CAMPO_CLAVE
GEDK925292	DESARROLLO6	RSII: Modificacion Programa de Copia de OrdenesCO
GEDK925278	DESARROLLO6	RSII-NO TRANSPORTAR!!!
GEDK925127	DESARROLLO6	RS II: Carga Maestro de CECOS.
GEDK925125	DESARROLLO6	RS II: Descarga de Activos Fijos
GEDK925123	DESARROLLO6	RS II: Descarga Maestro de Acreedores
GEDK925115	DESARROLLO6	RSII: Descarga Maestro de Clientes
GEDK924718	DESARROLLO6	RS - Menú de ámbito para plan de arranque
230	Copia de GEI(900) 260906	
-> GEI	Sistema GEI	
Modificable		
GEDK923749	DESARROLLO6	RS II-FI-Rep. ZFIICOMPPAR_TRANS v.4 181206

Se selecciona la orden que se quiere liberar y se pulsa el botón “Liberar directamente”. También se puede utilizar la tecla de función **F9**.

La transacción **STMS** se utiliza para transportar las órdenes de un mandante a otro. A continuación vemos su funcionamiento con las capturas de pantalla.

Ésta es la pantalla que se nos presenta al entrar en la transacción **STMS**. Pulsaremos sobre el botón  para al listado de los sistemas desde y a los que podemos transportar la orden.



Listado de sistemas:

Resumen import: Dominio DOMAIN_GEP

Ctd.colas import: 4 20.03.2007

Cola	Descripción	Órdenes
GED	System GED	51
GEI	Sistema GEI	408
GEP	System GEP	94
VIR	Sistema Virtual	13
		566

Seleccionamos el sistema al que queremos transportar nuestra orden, la que acabamos de liberar. En nuestro caso seleccionaremos GEI, visualizándose lo siguiente:

Cola import: Sistema GEI

Importar orden (Ctrl+F11)

Órdenes para GEI: 409 / 2425 21.03.2007 14:49:37

Número	Orden	Titular	Txt.breve	St
2394	GEDK927308	CVALERO01	GEOL_MM_OBYC nace11e mult1s 8842 8843 8942 8943_SD_20032007	✓
2395	GEDK927314	ACANALES01	Asignar transacción ZPP38 a dos roles	✓
2396	GEDK927316	IMARTINEZ01	GEOL FI Cambio en cuentas de diferencias T/C IM 200307	✓
2397	GEDK927318	HDIAZ01	GIT_CAT VAL PARA ZDUM Mait Fla_HD_20032007	✓
2398	GEDK927176	RARRONDO01	SD-6-322 nserie oblig. en entregas desde almacen con ubicac	✓
2399	GEDK927324	ABALLAZ01	GEOL_PU_Texto Mat Modif en ZI_AB_200307	✓
2400	GEDK927300	DESARROLLO6	Carga proveedores	■
2401	GEDK927288	RARRONDO01	Error al recepcionar directam. contra pedidos ZUB, LU, ZLI	✓
2402	GEDK927330	ARONCAL01	grupo imputación molde 652	✓
2403	GEDK924164	DESARROLLO4	R6M 04.01.2007 PMGAP11 Ajustes Idiomas	✓
2404	GEDK926868	RARRONDO01	Nota 931797	■
2405	GEDK927340	EVILLARON01	Actualización aprov especial centros 1520 y 4500	✓
2406	GEDK924563	DESARROLLO4	R6M 15.01.2007 PMGAP15B Estructura de Activos	✓
2407	GEDK927342	DESARROLLO2	D6S: Modificación bloqueo repartos	✓
2408	GEDK927338	DESARROLLO6	F2b: Programas de Descarga y Carga de Clientes y Proveedores	✓
2409	GEDK924487	DESARROLLO4	MML 12.01.2007 PMGAP15j Consulta horas trabajadas oper/subco	✓
2410	GEDK924537	DESARROLLO4	MFG 15.01.2007 PMGAP15M Materiales Serializables en la OT	✓
2411	GEDK925367	DESARROLLO4	MML 05.02.2007 PMGAP15k Consulta de DCAS y Alarmas	✓
2412	GEDK925544	DESARROLLO4	SRC 15.02.2007 PMGAP23 Check List	✓
2413	GEDK927344	IMARTINEZ01	GEOL FI Cuentas diferencias T/C IM 210307	✓
2414	GEDK927348	AMGRIJALVA01	GEOL MM AMG CREACION NUEVO ALMACEN 10040452 21.03.2007	✓
2415	GEDK927346	IMARTINEZ01	GEOL FI Cuentas dif T/C en revaluaciones IM 210307	✓

La orden que seleccionemos deberá estar pendiente de transportar, es decir con el símbolo . La seleccionamos y pulsamos el botón  que es el que transporta la orden al mandante destino que le indiquemos.

14.4 Análisis de errores (ST22)

La transacción ST22 permite visualizar los errores en tiempo de ejecución que se hayan producido desde un momento determinado a otro.

A continuación vemos el funcionamiento a través de las capturas de pantalla.

Al entrar en la transacción ST22 visualizaremos lo siguiente:

Errores tiempo ejecución ABAP

Ctd.errores tiempo ejecución.

Hoy	22
Ayer	21

Iniciar propia selec.

Selección propia

Fecha	21.03.2007	a		→
Hora	00:00:00	a	00:00:00	→
Máquina		a		→
Usuario	DESARROLLO6	a		→
Mandante		a		→
X= Guardar		a		→
ID de cancelación		a		→
Nom.programa (sólo ST22 nuev		a		→
EXcepción (sólo ST22 nueva)		a		→

Det.

Datos siguientes se determinan p.cada error tmpo.ejec.:

Programa afectado

Programa y compon.aplic.correspondientes (larga duración).

Se presentan dos opciones:

- Los errores de ejecución que se han producido hoy o ayer. Si pulsamos sobre cualquiera de los dos botones obtenemos un listado con los errores en tiempo de ejecución de hoy y ayer respectivamente. A la derecha de los botones se muestra un número que representa el número de errores.
- Otra opción es la de inspeccionar los errores que se han producido en un día determinado, a una hora específica, por un usuario concreto, etc. Para ello completaremos los campos del cuadro Selección propia, y pulsaremos el botón "Iniciar propia selección" para visualizar los errores.

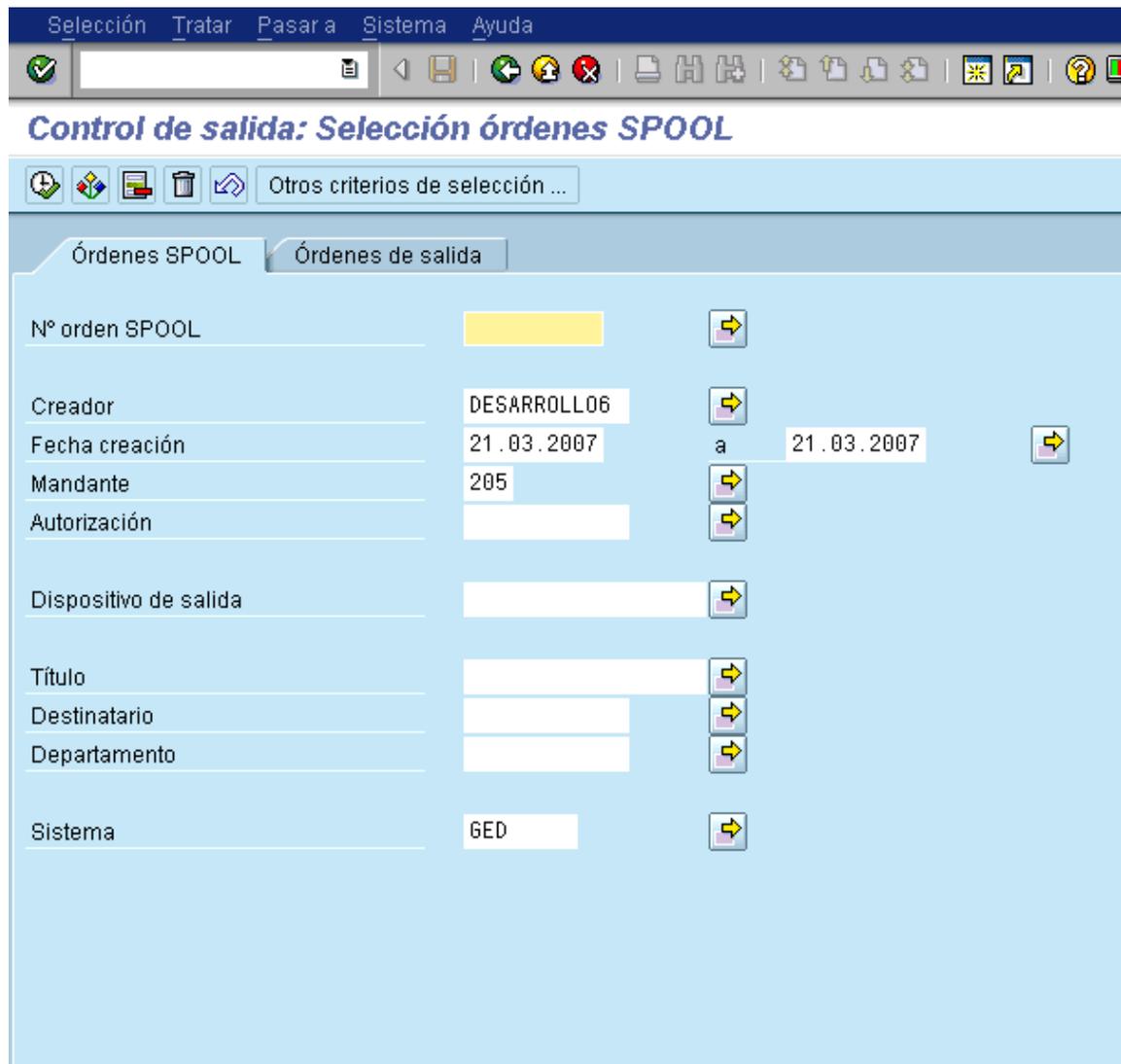
Ejemplo de listado de errores:

Listado de los errores de tiempo de ejecución seleccionados.

Fecha día	Hora	Máquina	Usuario	Md...	C	Nomb.err.tmpo.ejecución	Ex
21.03.2007	11:08:41	ged	DESARROLLO1	205	C	LOAD_PROGRAM_LOST	
21.03.2007	11:09:58	ged	DESARROLLO1	205	C	CALL_FUNCTION_CONFLICT_TYPE	
21.03.2007	11:43:31	ged	XLEJARRAGA01	230	C	LOAD_PROGRAM_LOST	
21.03.2007	11:49:01	ged	XLEJARRAGA01	230	C	LOAD_PROGRAM_LOST	
21.03.2007	11:56:21	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	11:56:45	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:00:22	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:00:46	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:00:56	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:41:04	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:41:22	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:41:32	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:43:00	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:44:28	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:44:44	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	12:45:59	ged	ARONCAL01	230	C	GETWA_NOT_ASSIGNED	
21.03.2007	13:06:31	ged	OMERONO01	230	C	LOAD_PROGRAM_LOST	
21.03.2007	13:15:34	ged	MMERIDA01	230	C	LOAD_PROGRAM_LOST	
21.03.2007	13:15:42	ged	MMERIDA01	230	C	LOAD_PROGRAM_LOST	
21.03.2007	13:16:13	ged	SDONAZAR01	215	C	OPEN_DATASET_NO_AUTHORITY	
21.03.2007	14:00:04	ged	DESARROLLO1	205	C	CONN_IMPORT_WRONG_COMP_LF	

14.5 Spool (SP01)

La transacción SP01 nos permite hacer una selección de las órdes de SPOOL. Para verlo, seleccionaremos la orden que queremos visualizar y pulsaremos el botón 



14.6 Programación de jobs

Tras haber definido un job utilizando la transacción SM36, podemos ejecutarlo mediante la transacción SM37. Esta transacción nos permite programar un job para su ejecución. La selección del job puede ser simple o ampliada.

Lo vemos a continuación en las capturas de pantalla.

Selección de job simple

 Ejecutar  Selección de job ampliada  Información

Job

Nombre de usuario

Status del job

Previs. Liberado Prep. Activo Termin. Cancelado

Condición de inicio de job

De  21.03.2007 A  21.03.2007

O tras evento

Paso de job

Programa ABAP

En el campo que está etiquetado con “Job”, deberemos especificar el job que vamos a ejecutar, es decir, el que vamos a programar.

En el cuadro “Condición de inicio del job”, informamos del momento en el que queremos que se ejecute nuestro job. Podemos proporcionar un rango de fechas y de horas. Si queremos que el job se ejecute sólo tras la ejecución de un evento, entonces deberemos seleccionar el evento en la opción “O tras evento”.

Podemos hacer una programación del job más ampliada. Para ello deberemos pulsar el botón “Selección de job ampliada”, con lo que se mostrará la siguiente pantalla.

Selección de job ampliada

Ejecutar Selección de job simple Información Reponer

Selección personal

Nombre selección

Job

Nombre de usuario

Destino ejecuc.

Servidor ejec.

Condición inic. Status Pasos Activo **Período**

Inicio previsto del job en el período

De A

O inicio en evento

O inicio tras job

O inicio en forma operación

Sugerencia de uso

Aquí aparecen más opciones para definir la programación del job. No sólo podremos seleccionar si el job se quiere ejecutar tras un evento determinado, sino que también se presenta la opción de ejecutarlo tras la ejecución de otro job o en forma de operación.

También se puede determinar si se quiere que el job se ejecute periódicamente. Para ello pulsaremos sobre la pestaña "Período". En ella encontraremos las siguientes opciones:

Condición inic. Status Pasos Activo **Período**

Período de job

Jobs (todos): Periódicos y no periódicos

Sólo jobs no periódicos

Sólo jobs periódicos

Se ejecutan 0 Meses

O cada 0 Sem.

O cada 0 Días

O cada 0 Horas

O cada 0 Minutos

Sugerencia de uso

De esta manera podemos determinar el período con el que queremos que se ejecute nuestro job.

14.7SXDA

14.8Editor split Screen (SE39)

El editor Split Screen (SE39), nos permite comparar programas. Es especialmente interesante cuando se emplea para comparar sistemas remotos, y poder determinar que los códigos fuente sean correctos.

Se presentan dos opciones:

- Comparación de programas dentro del mismo sistema:

Editor split screen ABAP: Imagen inicial

Comparación de sistemas

Fuentes a comparar

Izquierda

<input checked="" type="radio"/> Programa	Nombre	<input type="text"/>	<input checked="" type="radio"/> Por def.
<input type="radio"/> Método	Clase	<input type="text"/>	<input type="radio"/> Activa
	Método	<input type="text"/>	<input type="radio"/> Inactiva

Derecha

<input checked="" type="radio"/> Programa	Nombre	<input type="text"/>	<input type="radio"/> Por def.
<input type="radio"/> Método	Clase	<input type="text"/>	<input checked="" type="radio"/> Activa
	Método	<input type="text"/>	<input type="radio"/> Inactiva

 Visualizar  Modificar

Se deben especificar los programas que deben compararse, y no aparece opción para indicar el sistema.

- Comparación de programas que están en sistemas remotos (en integración y en desarrollo, por ejemplo). Para ello deberemos pulsar el botón “Comparación de sistemas”, mostrándonos la siguiente pantalla:

Editor split screen ABAP: Imagen inicial

Comparación dentro del sistema

Fuentes a comparar

Izquierda

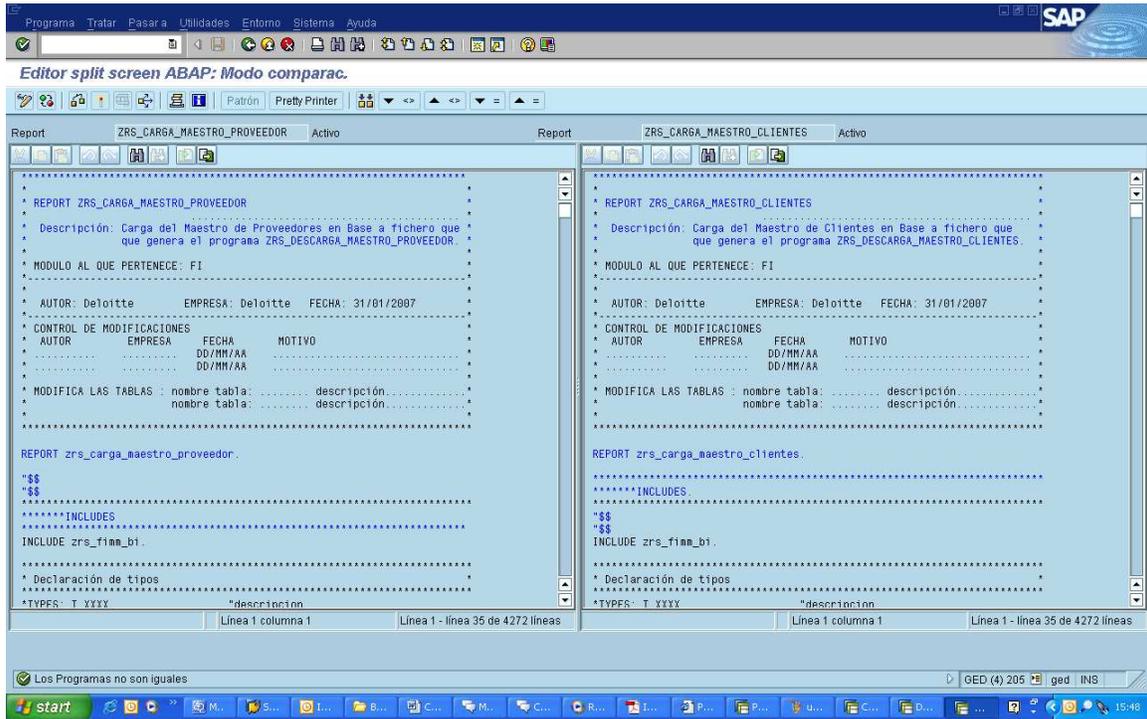
<input checked="" type="radio"/> Programa	Nombre	<input type="text" value=""/>	<input checked="" type="radio"/> Por def.
<input type="radio"/> Método	Clase	<input type="text" value=""/>	<input type="radio"/> Activa
	Método	<input type="text" value=""/>	<input type="radio"/> Inactiva
Sistema	GED		

Derecha

<input checked="" type="radio"/> Programa	Nombre	<input type="text" value=""/>	<input type="radio"/> Por def.
<input type="radio"/> Método	Clase	<input type="text" value=""/>	<input checked="" type="radio"/> Activa
	Método	<input type="text" value=""/>	<input type="radio"/> Inactiva
Destino RFC	<input type="text" value=""/>		

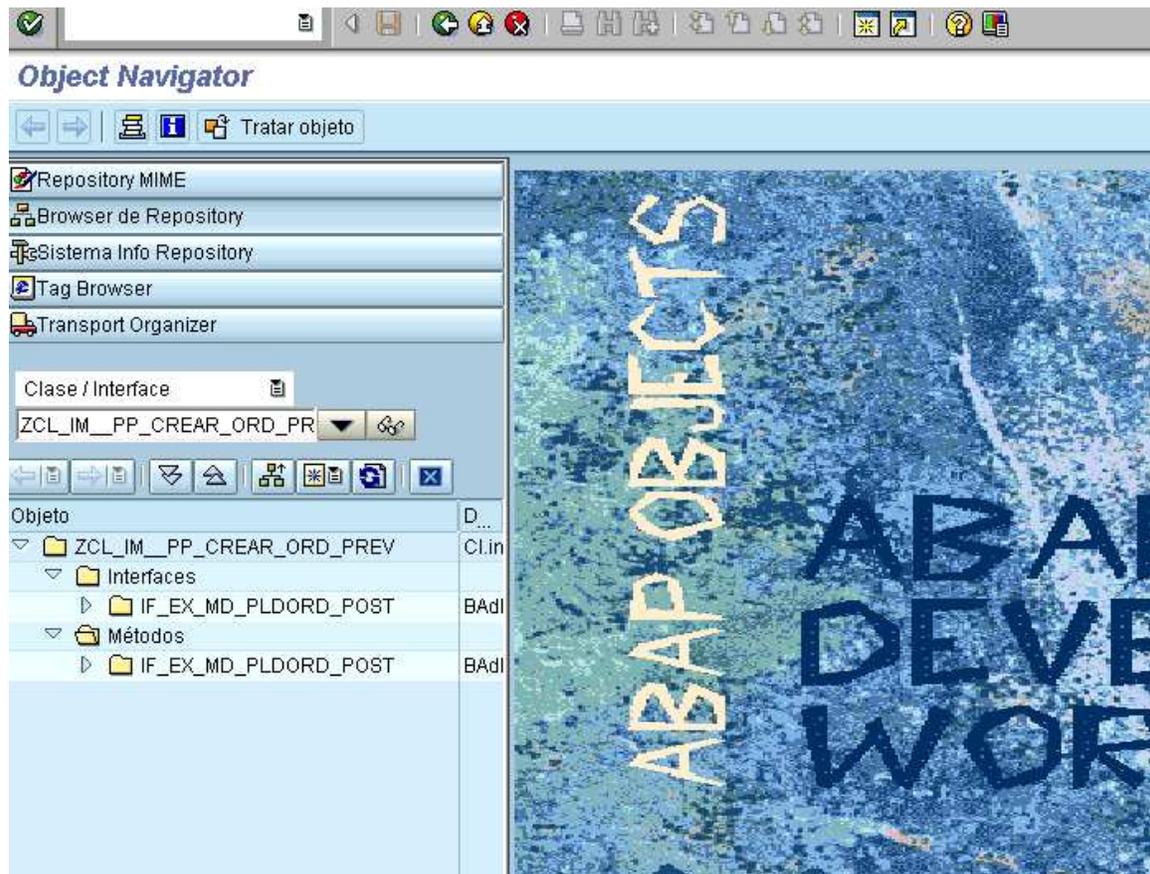
Compararemos el programa que se encuentra en el sistema en el que estamos en ese momento, con el que especifiquemos en el campo Destino RFC.

Al pulsar el botón visualizar obtendremos lo siguiente:



14.9 Object Navigator (SE80)

La transacción SE80 nos lleva a object navigator, que constituye un navegador de SAP. Nos permite ver y editar los objetos de SAP, packages, tablas, órdenes de transporte, programas, etc...



14.10 Menús de ámbito (SE43N)

La transacción SE43N, nos permite crear editar y visualizar un menú de ámbito. A continuación vamos a visualizar un menú de ámbito que ya ha sido creado:

El menú de ámbito **ZREESTRUCTURACION_SOCIETARIA**.

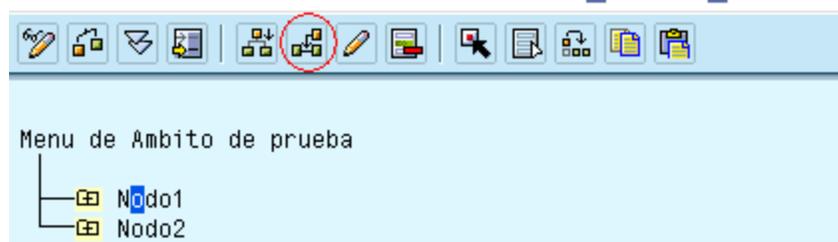
Visualizar menú de ámbito ZREESTRUCTURACION_SOCIETARIA



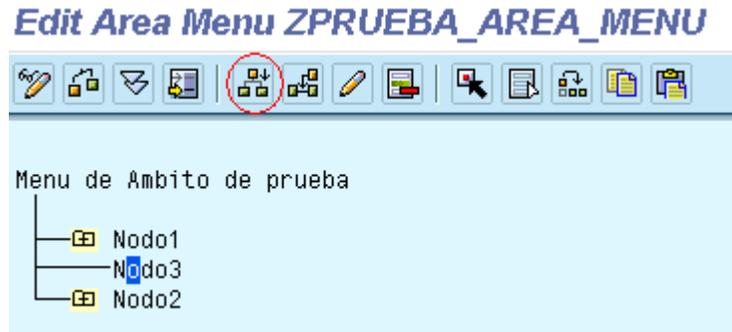
Este menú crea un enlace a cada uno de los programas que han de ejecutarse para realizar las cargas y descargas que se llevan a cabo en una reestructuración societaria

Para añadir un nodo a la misma altura del nodo raíz deberemos pulsar el botón , como se muestra a continuación:

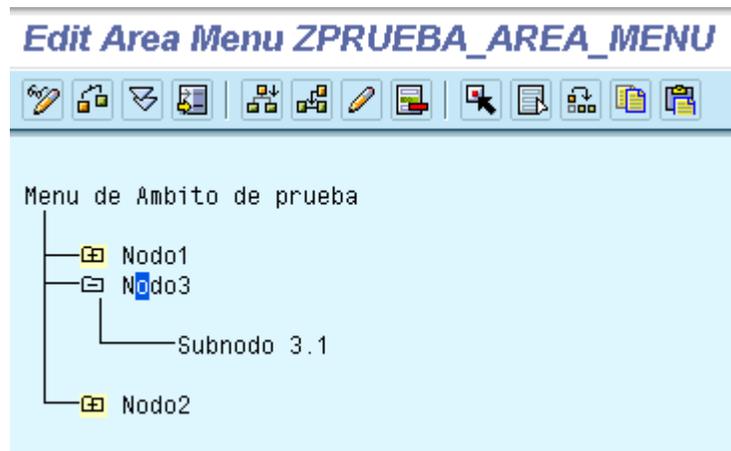
Tratar menú de ámbito ZPRUEBA_AREA_MENU



Y obtendremos lo siguiente:



Si queremos crear un subnodo deberemos pulsar el botón  , obteniendo lo que a continuación nos presenta la captura de pantalla:



Al pulsar cualquiera de los dos botones se nos mostrará una ventana en la que deberemos especificar el nombre que queremos que tenga el nodo o subnodo nuevo que queremos agregar al menú de ámbito.

14.11 Queries (sq01, sq02, sq03)

La transacción SQ01 nos sirve para definir las queries. A una query siempre se le asigna un grupo de usuarios que la va a utilizar, y un info set, en el que se determinan las tablas y la selección de campos de la query.

La transacción para definir una query es **SQ01**, y se muestra a continuación:

Curso programación ABAP IV

Query del grupo de usuarios /SAPQUERY/AB: Acceso

Queries del grupo usuarios /SAPQUERY/AB: Operación de agencia

Nombre	Título	InfoSet	Base de datos lógica	Tabla/Vista/Join	Título
AGENCY1	Operación de agencia: Nivel de documento individual	/SAPQUERY/AGENCY1	AGENCYLDB		Op
AGENCY2	Operación de agencia: Listas de documentos	/SAPQUERY/AGENCY2	AGENCYLDB		Op

La transacción **SQ02** permite definir el **info set**, la conexión de tablas y la selección de campos.

InfoSet: Acceso

Asignación a roles/gr.usu.

Estado	InfoSet	Descripción	Fuente de
■	/SAPQUERY/ACEDSACAC	ACAC Distribution Server InfoSet	Base de c
■	/SAPQUERY/ACEDSFILA	FILA Distribution Server InfoSet	Base de c
■	/SAPQUERY/ACEDSSOA	SOP Distribution Server InfoSet	Base de c
■	/SAPQUERY/ACEPSACAC	ACAC Posting Server InfoSet	Base de c
■	/SAPQUERY/ACEPSFILA	FILA Posting Server InfoSet	Base de c
■	/SAPQUERY/ACEPSSOA	SOP Posting Server InfoSet	Base de c

Podemos especificar un info set para modificarlo, visualizarlo o bien crearlo.

La transacción **SQ03** permite crear los grupos de usuarios que van a tener acceso a esa query.

Grupos de usuarios: Acceso

Área de trabajo Ámbito global (en todos los mandantes)

Grupo de usuario

Asignación a grupos de usuarios

Usuario

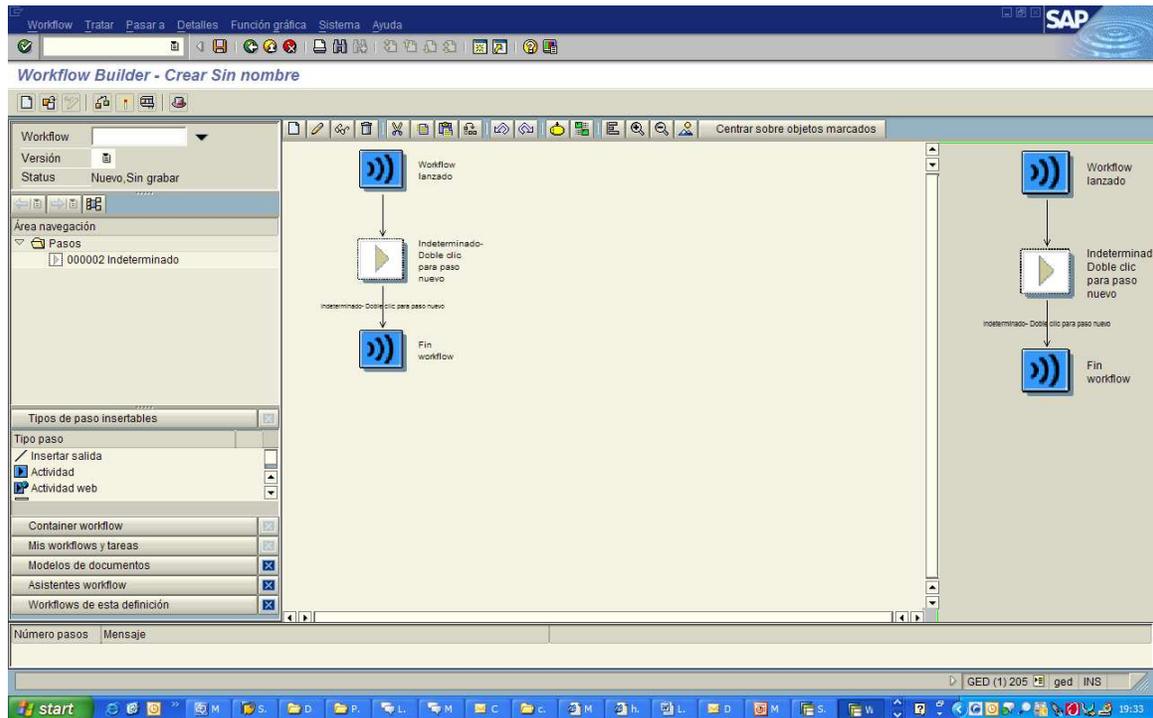
InfoSet

14.12 Workflow

Se trata de una herramienta que se utiliza para facilitar, automatizar y encadenar las tareas preestablecidas en los procedimientos de trabajo de cada usuario, es decir, automatiza los procesos de negocio de una empresa.

En concreto guía en la realización de una serie de tareas mediante mensajes en su Inbox a los usuarios autorizados para realizarlas.

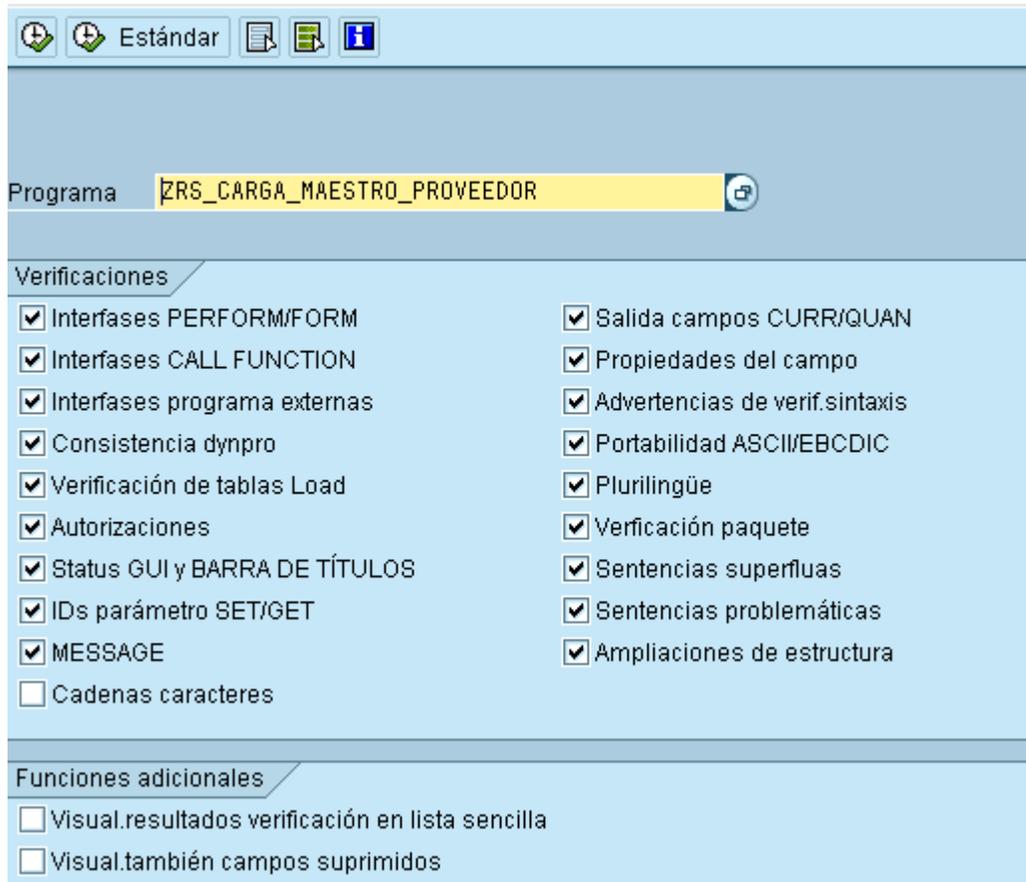
La transacción para editar WORKFLOWS es **SWDD**. La pantalla para crear WORKFLOWS y por tanto reflejar el flujo de actividades es la siguiente:



14.13 Verificación ampliada

La transacción es **SLIN**. Esta transacción nos permite verificar el programa de manera más exhaustiva. En este caso la verificación no se limita exclusivamente a los errores sintácticos, sino que además realiza las verificaciones que podemos ver detalladas en la siguiente captura de pantalla.

Verificación ampliada para un programa ABAP



Al pulsar el botón “Estándar” vemos un listado de los errores que el sistema ha encontrado. Muestra el número de errores que existen en el código para cada punto. Estos errores no son detectables cuando se verifica el código normalmente, es decir pulsando el botón  (verificación de código).

Resumen SLIN

<input type="button" value="Seleccionar"/> <input type="button" value="Vis.result.(todos)"/>			
Verificación programa ZRS_CARGA_MAESTRO_PROVE	Error	Advert.	Mensajes
Entorno de test	0	0	0
Interfases PERFORM/FORM	0	0	0
Interfases CALL FUNCTION	0	0	0
Interfases programa externas	0	0	0
Consistencia dynpro	0	0	0
Autorizaciones	0	0	0
Status GUI y BARRA DE TÍTULOS	0	0	0
IDs parámetro SET/GET	0	0	0
MENSAJE	0	0	0
Strings	12	0	0
Salida campos CURR/QUAN	0	0	0
Propiedades del campo	0	0	0
Sentencias superfluas	0	0	0
Advertencias de verif.sintaxis	0	0	0
Portabilidad ASCII/EBCDIC	0	0	0
Verificación del tamaño de carga	0	0	0
Plurilingüismo	0	0	0
Sentencias problemáticas	0	0	0
Ampliaciones de estructura	0	0	0
Verificación paquete	0	0	0
Errores y advertencias suprimidos	0	0	0