1. Ejecución condicionada de módulos

Si especificamos la cláusula **ON INPUT** después de MODULE en una instrucción FIELD, el módulo es ejecutado solamente si el campo en cuestión contiene un valor diferente al valor inicial.

En la sentencia CHAIN-­ENDCHAIN se debe usar la instrucción **ON CHAIN­-INPUT.** Entonces, el módulo es procesado solamente si al menos uno de los campos de la pantalla en la sentencia CHAIN­-ENDCHAIN contiene un valor diferente al valor inicial.

*Nota: Se puede usar la adición ON INPUT solamente si la instrucción MODULE es especificada dentro de una instrucción FIELD.*

Si se especifica la cláusula **ON REQUEST** después de MODULE en una instrucción FIELD, el módulo es ejecutado únicamente si el campo tiene una nueva entrada.

En la sentencia CHAIN­-ENDCHAIN, se debe usar la instrucción  **ON CHAIN-­REQUEST**. Entonces, el módulo concerniente es procesado solamente si al menos uno de los campos de pantalla de la sentencia CHAIN­ENDCHAIN tiene una nueva entrada.

*Nota: Se puede usar la adición* ***ON REQUEST*** *solamente si la instrucción MODULE es especificada dentro de una instrucción FIELD.*

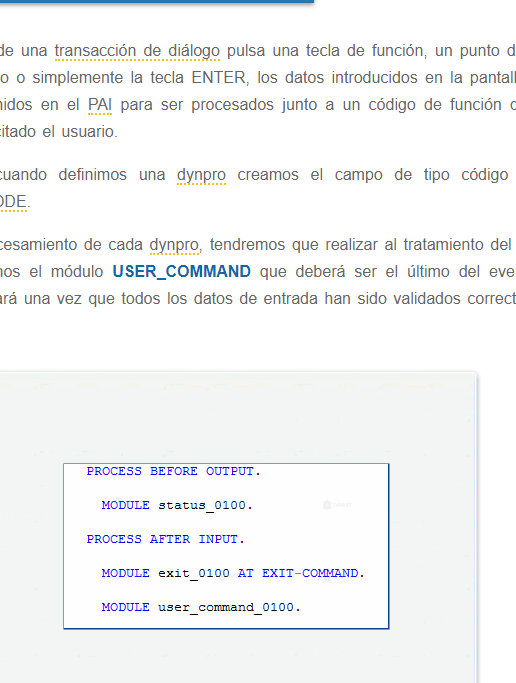
Utilizamos la cláusula AT EXIT-COMMAND de la instrucción MODULE para utilizar funciones estándar como BACK, EXIT, CANCEL, ejemplo: **MODULE <modulo\_ABAP> AT EXIT-COMMAND.**

Al utilizar el AT EXIT-COMMAND en un botón, es importante asignar el valor E en el atributo de campo TIPO FUNCIÓN. Para salir de la transacción o pantalla utilizamos por ejemplo: LEAVE TO SCREEN 0.

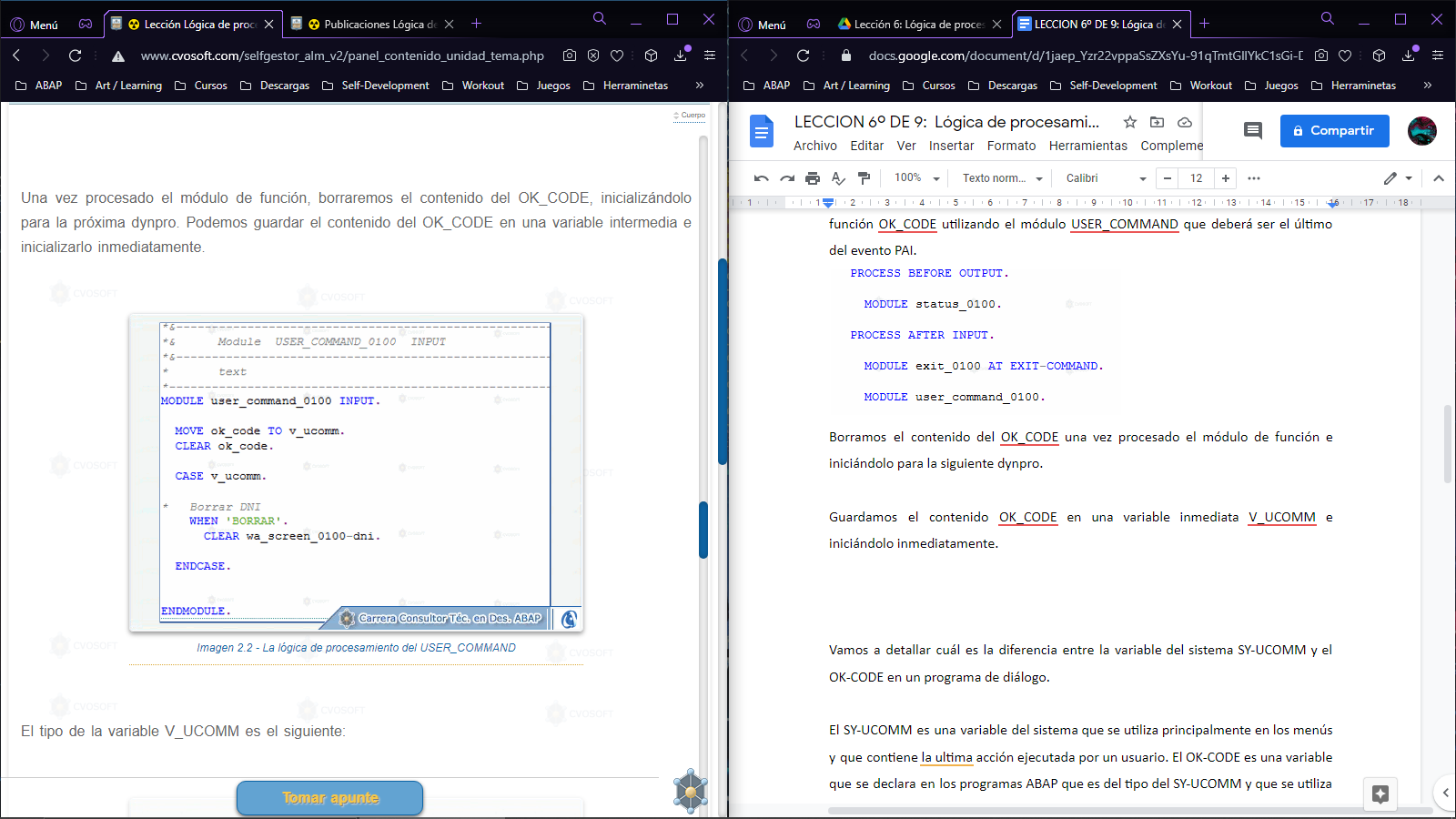
2. Tratamiento de los códigos de función

El usuario pulsa una tecla de función, un punto de menú, un pushbutton, un icono o simplemente ENTER, los datos introducidos pasan a los módulos definidos en el PAI para procesarlos junto a un código de función el cual indica la función solicitada por el usuario.

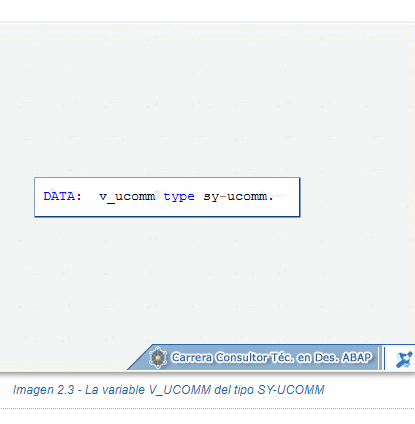
En cada lógica de procesamiento de la dynpro creamos el campo de código de función OK\_CODE utilizando el módulo USER\_COMMAND que deberá ser el último del evento PAI.



Borramos el contenido del OK\_CODE una vez procesado el módulo de función e iniciándolo para la siguiente dynpro.



Guardamos el contenido OK\_CODE en una variable inmediata V\_UCOMM e iniciándolo inmediatamente.



*Vamos a detallar cuál es la diferencia entre la variable del sistema* ***SY-UCOMM*** *y el* ***OK-CODE*** *en un programa de diálogo.*

*El* ***SY-UCOMM*** *es una variable del sistema que se utiliza principalmente en los* ***menús*** *y que* ***contiene la última acción ejecutada por un usuario****. El* ***OK-CODE*** *es una variable que se declara en los programas ABAP que es del tipo del SY-UCOMM y que se utiliza generalmente* ***en las pantallas*** *el OK-CODE actúa solamente como una* ***variable temporal que almacena el valor de SY-UCOMM****. Cuando el usuario interactúa con los elementos de la pantalla el código de función asignado rellena la variable SY-UCOMM que a su vez se refleja en el OK-CODE en nuestros programas ABAP debemos trabajar con el OK-CODE en lugar del SY-UCOMM.*

*Existen dos razones para esto primeramente el programa ABAP tiene el control total sobre los campos declarados en él y en segundo lugar* ***nunca se debe cambiar el valor de una variable del sistema ABAP****. Sin embargo siempre debemos inicializar el campo OK-CODE en los programas ABAP debido a que, de la misma manera que el OK-CODE y el SY-UCOMM reciben el contenido de los campos de la pantalla correspondientes, en el caso del PAI su contenido se asigna al campo de la pantalla OK-CODE y a la variable del sistema SY-UCOMM en el PBO, por lo tanto se debe limpiar el OK-CODE en programa ABAP para asegurarnos que el código de función de una pantalla no este lleno en el PBO por un valor no deseado.*

*Esto es particularmente importante cuando el próximo evento del PAI se pueda activar con un código de función vacío por ejemplo presionando la tecla enter.*

3. Secuencias dinámica de las pantallas.

En un programa de diálogo o también llamado transacción de diálogo podemos controlar la secuencia de ejecución de cada una de las dynpros que componen a la transacción.

Existen dos instrucciones que nos permitirán pasar a otra dynpro existente dentro del mismo programa. La primera de ellas es la instrucción **SET SCREEN** que tiene la siguiente sintaxis: **SET SCREEN <nro\_pantalla>.**

La instrucción SET SCREEN reescribe temporalmente la siguiente pantalla a procesar. La pantalla siguiente debe ser una pantalla del mismo Module Pool.

**La pantalla siguiente es procesada después de procesar la pantalla actual** o al menos que se termina la ejecución de la pantalla actual con la instrucción **LEAVE SCREEN**. Al encontrar esta instrucción, **se ejecuta la pantalla siguiente en forma inmediata**.

Si se desea **terminar el procesamiento de la pantalla actual e ir directamente a la pantalla siguiente** en una sola instrucción, se puede usar la sentencia **LEAVE TO SCREEN** tal como vemos a continuación.

**LEAVE TO SCREEN <nro\_pantalla>.**

La instrucción **CALL SCREEN interrumpe el procesamiento de la pantalla actual** para procesar la pantalla X y las pantallas subsecuentes. CALL SCREEN 0200.

Cualquiera de las instrucciones: **SET SCREEN 0, LEAVE SCREEN, LEAVE TO SCREEN 0,** **regresa el control al lugar donde fue ejecutada la instrucción CALL SCREEN.**

Si se usa cualquiera de las instrucciones anteriores cuando no se está en el modo de llamada, es decir cuando no se cedió el control a otra dynpro del programa, entonces el programa termina.

Usando las cláusulas **STARTING AT** y  **ENDING AT** en la instrucción **CALL SCREEN**, se puede **especificar la posición y el tamaño de la pantalla a llamar.** Veamos el siguiente ejemplo:

CALL SCREEN 0200 STARTING AT 5 1 ENDING AT 130 25.

Si la pantalla aparece incompleta, se incluye una barra de desplazamiento.