UNIDAD

7

DIPLOMATURA EN PROGRAMACION ABAP

MÓDULO 7: EVENTOS

EVENTOS

Este módulo introduce las dos últimas de las cinco unidades de modularización que posee ABAP/4: eventos y funciones. La primera se describe en detalle. De las funciones se lleva a cabo una mera introducción conceptual, dado que se desarrollará el tema en detalle en la Unidad 9.

Concepto de Evento

Contrariamente a las primeras versiones de ABAP/4 y a lo que se puede presuponer, los programas de ABAP / 4 son orientados y manejados por eventos. Una buena comprensión de los eventos es entonces la clave para una buena comprensión de ABAP / 4.

Definimos *evento* como una etiqueta que identifica una sección de código. La sección de código asociado a un evento comienza con un nombre de evento y termina cuando se encuentra el nombre del siguiente evento. En el listado a continuación, los nombres de los eventos son: `initialization`, `start-of-selection` y `end-of-selection`:

```
1  report ztx0701.
2  initialization.
3    write / '1'.
4
5  start-of-selection.
6    write / '2'.
7
8  end-of-selection.
9    write / '3'.
```

Los nombres de eventos son palabras reservadas. Usted no puede crear nuevos eventos, sólo puede usar los ya existentes.

El código del listado anterior genera el siguiente resultado:

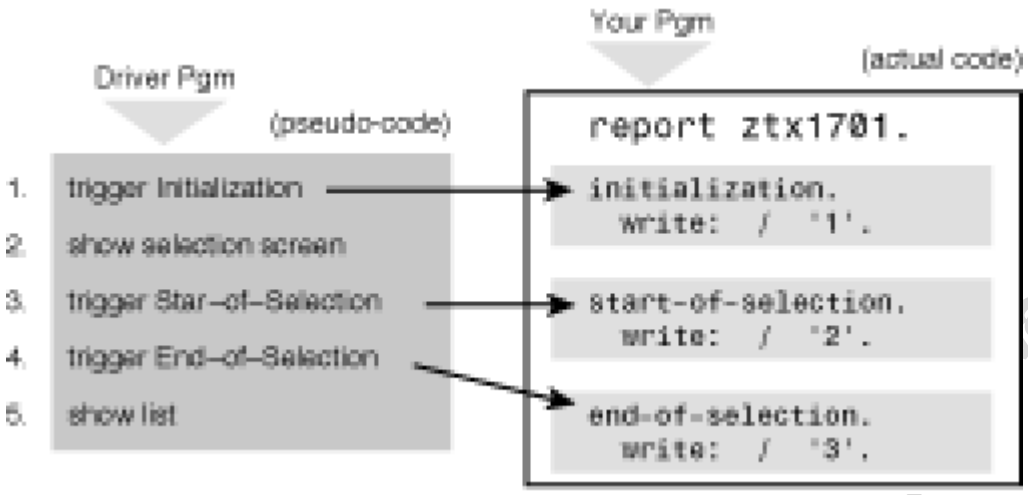
```
1
2
3
```

Podemos analizarlo de la siguiente manera:

- La línea 2 identifica el comienzo del evento `initialization`.
- La línea 3 se asocia el evento `initialization`. Si hay más líneas de código luego de ésta, serían todas pertenecientes al evento `initialization`.
- La línea 5 marca el final del código que pertenece al evento `initialization` y el comienzo del evento `start-of-selection` .
- La línea 6 pertenece al evento `start-of-selection`.
- La línea 8 marca el límite inferior del código del evento `start-of-selection` y el límite superior del evento `end-of-selection`.
- La línea 9 pertenece al evento `end-of-selection`.
- El límite inferior de la código perteneciente al evento `end-of-selection` está marcado por el final del programa.
- Cuando se ejecuta este programa, estos eventos son disparados por un programa controlador. Los siguientes párrafos explicarán este concepto en detalle.

Un programa controlador o *driver* es un programa que controla otro programa (controlado o *driven*). SAP suministra programas controladores con el sistema R / 3. Usted suministra (escribe) el programa controlado . Al iniciar su programa, un programa controlador se inicia siempre en primer lugar, y luego se llama a los eventos en su programa. Para reiterar, un programa *driver* se inicia primero y luego controla cuándo su programa tomará el control. Esto siempre ha sido el caso de todos los programas que ha escrito hasta ahora, simplemente no ha sido consciente de ello hasta ahora. Por favor, asegúrese de leer este párrafo con cuidado. Para repetir una vez más, **cuando usted comience el programa, un programa *driver* se inicia primero y controla el programa llamando a los eventos dentro del mismo.**

El código asociado a un evento se desencadena por una declaración en un programa *driver*. Los eventos se desencadenan por el programa *driver* en una secuencia predefinida y predecible. La figura siguiente ilustra este punto:



La figura contiene el pseudocódigo dentro del programa piloto o *driver*. El código que se muestra en el programa de la derecha es el código real. Cuando el programa se ejecuta, el controlador de la izquierda comienza primero. Entonces, el programa sigue esta secuencia de pasos:

- Se desencadena el evento `initialization`, haciendo que el código que pertenece a `initialization` sea ejecutado. Si usted no ha codificado un evento `initialization` en su programa, el programa piloto se salta este paso.
- Se muestra la pantalla de selección para el programa. (Una pantalla de selección es la pantalla que contiene los campos de entrada para sus parámetros de usuario, si existieran.) Si su programa no tiene una pantalla de selección, se salta este paso.
- Se dispara el evento `start-of-selection`, haciendo que el código que pertenece a ese evento pase a ser ejecutado. Si usted no ha codificado un evento `start-of-selection` en su programa, se salta este paso.
- Se dispara el evento `end-of-selection` en su programa. Si usted no ha codificado un evento `end-of-selection` , se salta este paso.
- Se muestra la lista de salida para el usuario.

El orden de ejecución de eventos se determina por el programa controlador, no por su programa. Por lo tanto, se puede codificar los eventos en cualquier orden y el orden de ejecución seguirá siendo el mismo. El orden de los eventos en su programa no tiene importancia, sino que siempre se iniciará en la misma secuencia en el programa piloto. El siguiente listado ilustra este punto más acabadamente:

```
1  report ztx0702.
2  data f1 type i value 1.
3
4  end-of-selection.
5    write: / '3.  f1 =', f1.
6
7  start-of-selection.
8    write: / '2.  f1 =', f1.
9    f1 = 99.
10
11 initialization.
12   write: / '1.  f1 =', f1.
13   add 1 to f1.
```

El código del listado anterior genera el siguiente resultado:

```
1. f1 = 1
2. f1 = 2
3. f1 = 99
```

Podemos analizarlo de la siguiente manera:

- Al iniciar `ztx0702` , el programa piloto se inicia en primer lugar.
- Se desencadena el evento `initialization`.
- El código asociado con el evento `initialization` se ejecuta (líneas 12 y 13). El valor de `f1` se escribe y se añade *1* a la salida. El control vuelve entonces al programa controlador.
- El programa piloto busca una pantalla de selección. No hay una para este programa, por lo se pasa al evento `start-of-selection`. Se encuentra en la línea 7, por lo que las líneas 8 y 9 se ejecutan y el control después vuelve al controlador.
- El programa piloto busca un evento `end-of-selection` entonces. Al encontrarlo en la línea 4, se transfiere el control a `ztx1702` que comienza a ejecutarse en la línea 5, y el control vuelve entonces al conductor o *driver*.
- El conductor muestra la lista al usuario.

Este ejemplo ilustra que usted puede poner los hechos en un orden diferente, pero la secuencia de ejecución no cambia. La secuencia de ejecución es siempre `initialization`, `start-of-selection`, `end-of-selection`. Hay otros eventos existentes en ABAP/4, algunos se producen después de la inicialización y algunos se producen entre los de inicio de la selección y de fin de la selección . Los veremos más adelante

Los programadores, a pesar de lo aclarado antes, generalmente posicionan por convención los eventos en el orden en el que se activan, para hacer que sus programas más fácil de entender.

En la siguiente tabla se muestran el top 10 de eventos de ABAP (los más usados), categorizados por quien los dispara (programa controlador, interacción de usuario o programa de usuario):

Categorías	Eventos

Driver	`initialization`
	`at selection-screen`
	`start-of-selection`
	`get`
	`end-of-selection`

Usuario	`at line-selection`
	`at pfn`
	`at user-command`

Programa	`top-of-page` `end-of-page`

Los de la primera categoría son disparados por el programa controlador. Los eventos de usuario son disparados por el usuario a través de la interfaz de usuario. Los de programa son los provocados desde el programa de usuario.

En la ayuda de SAP puede encontrarse más material sobre estos y otros eventos. En el resto del curso nos centraremos no sólo en los tres eventos recién descriptos, sino que también iremos viendo el resto a medida que vayan apareciendo en relación a otros temas.

Sigamos analizando ahora los tres eventos de *driver*.

Si la primera instrucción ejecutable en su programa no está precedido por un nombre de evento, en tiempo de ejecución el sistema inserta automáticamente `start-of-selection` antes de la primera línea de código ejecutable. Por lo tanto, si no se ha codificado ningún evento, o usted ha puesto el código en la parte superior del programa sin nombrar a un evento, se asume como "evento por *default*" `start-of-selection`.

De hecho, todos los programas que hemos visto en capítulos anteriores y que no tenían codificado ningún evento, en realidad estaban siempre ejecutando el evento `start-of-selection`.

El siguiente listado proporciona un ejemplo de esto, en un programa que contiene tanto una pantalla como eventos de selección. También contiene un evento de programa impulsado por el evento de inicio de página, `top-of-page`, que se verá en detalle en la próxima unidad:

```
1  report ztx0703 no standard page heading.
2  parameters p1(8).
3
4  write: / 'p1 =', p1.
5
6  initialization.
7    p1 = 'INICIO'.
8
9  end-of-selection.
10   write: /(14) sy-uline,
11          / 'Fin'.
12
13 top-of-page.
14   write: / 'Título'.
15   skip.
```

El código del listado anterior genera el siguiente resultado:

```
Título
p1 = INICIO
--------------
Fin
```

Podemos analizarlo de la siguiente manera:

- Al ejecutar el programa `ztx0703` , un programa piloto se inicia en primer lugar. Dado que no hay código al principio del programa y que no pertenece a ningún evento, `start-of-selection` se inserta automáticamente en la línea 3.
- El programa controlador desencadena el evento initialization. La línea 7 se ejecuta y se asigna el valor `'INICIO'` a `p1` . El control vuelve al programa piloto.
- Puesto que hay una pantalla de selección para este programa, ahora se muestra la pantalla de selección. El valor asignado a `p1` en la línea 7 aparece en el campo de entrada
- El usuario pulsa Ejecutar (*F8*) en la pantalla de selección. El control vuelve al programa controlador, lo cual dispara el evento `start-of-selection`.
- Dado que `start-of-selection` se insertó en la línea 3, el control pasa a `ztx0703` a partir de la línea 4.
- La línea 4 se ejecuta. Dado que esta es la primera sentencia `write` ejecutada después de `start-of-selection`, se dispara el evento `top-of-page`. Su funcionamiento se explicará en detalle en la próxima unidad. Se transfiere el control a la línea 14.
- La línea 14 escribe un título. La línea 15, escribe una línea en blanco. El control vuelve a la línea 4.

- La línea 4 escribe el valor de `p1` .
- El conductor activa el evento `end-of-selection`. El control pasa a la línea 10.
- La línea 10 escribe una línea horizontal de 14 caracteres (estas opciones de formato de salida se explicarán en detalle en la próxima unidad).
- La línea 11 escribe `Fin` . El control vuelve al programa piloto.
- El conductor muestra la lista al usuario.

No se puede poner una condición en torno a un evento o adjuntar un evento en un bucle. Ello provocaría un error de sintaxis. Por ejemplo, el código del listado siguiente genera un error de sintaxis.

```
1  report ztx0704.
2  data f1.
3
4  start-of-selection.
5    f1 = 'A'.
6
7  if f1 = 'A'.
8    end-of-selection.
9    write: / f1.
10   endif.
```

El código del listado anterior genera este error de sintaxis

```
Incorrect nesting: before the statement "END-OF-SELECTION", the structure
introduced by "IF" must be concluded by "ENDIF".
```

Es decir, que la anidación es incorrecta antes de la declaración end-of-selection, la estructura introducida por "IF" debe ser concluida por "ENDIF".

Analicémoslo en detalle:

- Los nombres de evento tienen mayor prioridad que otras declaraciones ABAP / 4. El `if` en la línea 7 pertenece al evento `start-of-selection`. La declaración en la línea 10 `endif` pertenece al evento `end-of-selection`. Se deben cerrar todas las condiciones y bucles abiertos dentro del evento al que pertenecen. Por lo tanto, este programa debe tener un `endif` antes que el evento `end-of-selection` . Debido a que existe también un `endif` dentro de `end-of-selection` en la línea 10, este `endif` tendría que ser eliminado antes de que el programa se ejecute.
- Usted no debe poner las definiciones de datos dentro de los eventos. Aunque esto no causa un error de sintaxis, es un estilo de programación pobre y no convencional. Todas las definiciones de datos se

deben hacer en la parte superior del programa (como, en este caso, en la línea 2).

- Cada evento tiene un propósito específico y se necesita para llevar a cabo una tarea de programación específica. A medida que surja la necesidad, me referiré a ellos en el material que sigue. Si no se ven eventos en un programa, recuerde siempre que siempre existe en forma implícita por *default* el evento `start-of-selection`.

Para abandonar un evento en cualquier momento con las siguientes declaraciones:

- `exit`
- `check`
- `stop`

Los siguientes párrafos describen el efecto de `check` y de `exit` cuando se codifican fuera de un bucle. El efecto de `stop` es el mismo independientemente de si se es codificado dentro de un bucle o no. La sentencia `check` aparece en otras unidades relacionado a otros temas de ABAP/4, sobre todo para el manejo de sentencias de control dentro de bucles.

En **todos** los eventos estas declaraciones funcionan de la siguiente manera:

- `check` deja inmediatamente el evento actual y el procesamiento continúa con el siguiente evento (o acción, como mostrar la pantalla de selección o lista de salida).
- `stop` deja inmediatamente el evento actual y va directamente al evento `end-of-selection`. La ejecución de `stop` dentro de end-of-selection deja el evento. No causa un bucle infinito.

En los eventos que se producen antes de `start-of-selection`:

- `exit` y `check` tienen el mismo comportamiento. Ambos dejan el evento inmediatamente y el proceso continúa con el siguiente evento (o acción, como la visualización de la pantalla de selección).

En `start-of-selection` y en los eventos que se producen después del mismo:

- `exit` termina el informe y muestra la lista de salida. Existe una sola excepción, dentro de `top-of-page` , `exit` se comporta como `check`.
- `check` deja el evento y el proceso continúa con el siguiente evento (o acción, como la pantalla de la lista de salida).

No utilice `stop` en los siguientes eventos: `initialization`, `at selection-screen output`, `top-of-page` y `end-of-page`. Técnicamente, `stop` puede trabajar con `top-of-page` y `end-of-page`, si usted se abstiene de emitir declaraciones `write` dentro de `end-of-selection` después. En el caso de `top-of-page`, `write` puede causar un volcado (*short dump*) y en el caso del `end-f-page`, se puede perder de salida. Es más seguro para evitar por completo dentro de estos eventos.

`check`, `exit` y `stop` *no* establecen el valor de `sy-SUBRC` . Si desea configurarla, puede asignar un valor numérico a la variable `sy-SUBRC` antes de salir del evento

El informe que se muestra a continuación le permite probar los efectos de estas declaraciones dentro de diversos eventos.

Copie y quite los comentarios de uno en uno para experimentar las variaciones posibles, a partir de la línea 24:

```
1  report ztx0705 no standard page heading line-count 6(2).
2 * en los eventos antes de start-of-selection:
3 * - exit y check tienen el mismo comportamiento. Ambos dejan el evento
4 * y el proceso continúa con el próximo evento o acción.
5 * - stop va directamente al evento end-of-selection
6 * (no utilice stop en initialization o at selection-screen output)
7
8 * en start-of-selection y los eventos posteriores:
9 * - exit termina el informe y muestra la lista de salida
10 * excepción: en top-of-page, exit sale del evento
11 * - check deja el evento y el proceso continúa con el siguiente.
12 * - stop va directamente al evento end-of-selection
13
14 parameters: "                    ejecutar un:
15 exit_sos radioButton group g1 "exit en el start-of-selection
16 exit_eos radioButton group g1 "exit en el end-of-selection
17 chck_sos radioButton group g1 "check en start-of-selection
18 chck_eos radioButton group g1 "check en end-of-selection
19 stop_sos radioButton group g1, "stop  en start-of-selection
20 stop_eos radioButton group g1, "stop en el end-of-selection
21 none radioButton group g1. "ninguno de los tres
22
23 initialization.
24 *   exit.                          "sale del evento
25 *   check 1 = 2.                  "es falso siempre, sale del evento
26 *   stop.                          "no use stop en initialization
27     chck_sos = 'X'.
28
29 at selection-screen output.
30 *   exit.                          " sale del evento
31 *   check 1 = 2.                   " sale del evento
32 *   stop.                          " no use stop en este evento
33     message s789(zk) with 'at selection-screen output'.
34
35 at selection-screen on radiobutton group g1.
36 *   exit.                          " sale del evento
37 *   check 1 = 2.                   " sale del evento
38 *   stop.                          " va a end-of-selection
39     message i789(zk) with 'at selection-screen on rbg'.
40
41 at selection-screen.
42 *   exit.                          " sale del evento
43 *   check 1 = 2.                   " sale del evento
44 *   stop.                          "va a end-of-selection
45     message i789(zk) with 'at selection-screen'.
46
47 start-of-selection.
48     write: / 'Comienzo de Start-of-Selection'.
49     if      exit_sos = 'X'.
50         exit.                      " sale del programa
51     elseif  chck_sos = 'X'.
52         check 1 = 2.               "sale del programa
53     elseif  stop_sos = 'X'.
54         stop.                      "va a end-of-selection
```

```
55          endif.
56      write: / 'Fin de Start-of-Selection'.
57
58  end-of-selection.
59      write: / 'Comienzo de End-of-Selection'.
60      if      exit_eos = 'X'.
61          exit.                       " sale del programa
62      elseif  chck_eos = 'X'.
63          check 1 = 2.                " sale del programa
64      elseif  stop_eos = 'X'.
65          stop.                       " sale del programa
66          endif.
67      write: / 'Fin de End-of-Selection'.
68      write: / '1',
69             / '2',
70             / '3'.
71
72  top-of-page.
73      write: / 'Título'.
74  *   exit.            "sale del evento y vuelve a la sentencia write
75  *   check 'X' = 'Y'. "sale del evento y vuelve a la sentencia write
76  *   stop.              "va a end-of-selection y no escribe nada más
77      uline.
78
79  end-of-page.
80      uline.
81  *   exit.                    " sale del programa
82  *   check 'X' = 'Y'. "sale del evento y vuelve a la sentencia write
83  *   stop.          "va a end-of-selection y no escribe nada más
84      write: / 'Final'.
```

Si bien la corrida de este programa se plantea como ejercicio en el material práctico del curso, apuntemos algunos detalles a modo de somero análisis:

- Las líneas 15 y siguientes define un grupo de botones de radio. La elección de uno de ellos hará que la declaración mencionada se ejecute como se indica en el comentario de cada una de las líneas. Por ejemplo, la elección de `exit_sos` hace que el comando `exit` pase a ejecutarse en el evento `start-of-selection` y así siguiendo para todos los demás.
- `check 1 = 2` está *hardcodeado* (no es modificable) por lo que siempre devuelve un resultado negativo.

Echemos un vistazo más de cerca a los eventos en el contexto de un informe que tiene una pantalla de selección. (aquellos que tienen, como el anterior, una sentencia *parameters*, para permitir que el usuario ingrese datos en una pantalla previa a la aparición del listado final.) y veamos lo que sucede cuando uno vuelve hacia atrás luego de ver el listado final.

Cuando el usuario ejecuta el informe, el *driver* dispara `initialization` y luego se muestra la pantalla de selección. Después de pulsar el botón Ejecutar (F8), el *driver* activa los eventos restantes, el programa termina y el usuario ve la lista. El usuario pulsa el botón Atrás (F3) para volver. El *driver* reinicia el tratamiento comenzando en la parte superior de la lista de eventos. Se desencadena el evento `initialization` , y luego todos los eventos subsiguientes siguen de nuevo en su secuencia normal. El resultado es que el usuario ve la pantalla de selección después de pulsar el botón Atrás. Hay, sin embargo, una diferencia en el procesamiento cuando se produce un reinicio.

La pantalla de selección tiene su propia copia de todas las variables que se muestran en ella. La primera vez que se ejecuta el informe y el programa *driver* recupera el control después de que el evento `initialization` ya ha finalizado, se copian los valores de las variables del programa a las variables de la pantalla de selección correspondientes y se los muestra. El usuario puede modificar los campos de entrada. Cuando el usuario presiona el botón Ejecutar, el controlador almacena los valores de la pantalla de selección en dos áreas de datos: uno perteneciente a la pantalla de selección y luego en las variables de su programa de usuario.

Sin embargo, esta doble acción sólo se produce la primera vez que se ejecuta el programa. Cuando el usuario presiona Atrás, se activa de nuevo `initialization` . Cuando el control vuelve al conductor, el mismo no copia las variables del programa en el área de datos de la pantalla (*screen data area*). En su lugar, se muestran los valores actuales de la zona de datos de la pantalla, que todavía contiene los valores que el usuario había introducido. El resultado es que el usuario puede ver los valores que entró la última vez, sin tener en cuenta lo que ha cambiado en su programa. Entonces, después de que el usuario presione el botón Ejecutar, los valores en pantalla se copian en el área de datos de la pantalla y luego a su programa, sobrescribiendo cualquier diferencia. Por ejemplo, si configura valores en `initialization`, los valores se pueden ver en la pantalla de selección cuando se inicia el programa. Al pulsar el botón Atrás en la pantalla de selección, el evento `initialization` se ejecutará pero los valores fijados en su interior no se mostrarán luego. El usuario en cambio seguirá viendo los mismos valores que ha introducido.

En resumen, podemos decir que:

- Existen eventos en todos los programas.
- Si ninguno se codifica, el evento predeterminado `start-of-selection` se inserta automáticamente.

- `initialization` se utiliza a menudo para asignar valores a los parámetros que aparecen en una pantalla de selección.
- `end-of-selection` es para la limpieza del programa, los resúmenes, o las rutinas de terminación anormal.
- Los eventos se desencadenan por parte el programa piloto o *driver*. El conductor siempre se inicia antes que su programa y controla cuándo se ejecutan los eventos.
- Usted no puede colocar una condición en torno a un nombre de evento.
- Todos los eventos se activan de nuevo cuando el usuario regresa de la lista. Sin embargo, las variables de la pantalla de selección no se recargan ni actualizan desde el área de datos del programa.
- Use `exit`, `check` o `stop` para abandonar un evento.

Introducción a los módulos de funciones

Veremos a continuación los conceptos y características básicas de los módulos de funciones en ABAP/4. En la Unidad 9 retomaremos este tema para verlo en profundidad, tratando sus características avanzadas.

Un *módulo de función* es la última de las cinco unidades principales de modularización de ABAP / 4.

En principio, es muy similar a una subrutina externa de las siguientes maneras:

- Ambas existen dentro de un programa externo.
- Ambas permiten los parámetros ser pasados y devueltos.
- Los parámetros se pueden pasar por valor, por su valor y resultado, o por referencia.

Las principales diferencias entre los módulos de funciones y subrutinas externas son las siguientes:

- Los módulos de función tienen una pantalla especial que se utiliza para definir los parámetros (los parámetros no están definidos a través de declaraciones ABAP / 4).
- Las áreas de trabajo `tables` *no* se comparten entre el módulo de función y el programa de llamada.
- Se utiliza diferente sintaxis para llamar a un módulo de función que la que se usa para llamar a una subrutina.
- Dejar un módulo de función se realiza por medio de la declaración `raise` en lugar de `check` , `exit` , o `stop` .

Un nombre de módulo de función tiene una longitud mínima práctica de tres caracteres y una longitud máxima de 30 caracteres. Los nombres de módulos de función del cliente deben comenzar con *Y*_ o *Z*_ . El nombre de cada módulo de función es único dentro de todo el sistema R / 3.