UNIDAD

9

DIPLOMATURA EN PROGRAMACION ABAP

MÓDULO 9: FUNCIONES

FUNCIONES

Este módulo introduce los conceptos relacionados con el uso detallado de funciones en ABAP.

Grupos de funciones

Continuando con el tema de funciones introducido en la Unidad 7, diremos ahora que un grupo de funciones es un programa que contiene módulos de función. A partir de este momento *función* y *módulo de función* serán tratados como términos indistintos y equivalentes.

Con cada sistema R / 3, SAP suministra más de 5.000 grupos de funciones pre-existentes. En total, contiene más de 30.000 módulos de función. Si requiere alguna funcionalidad que no está cubierto por estos módulos funcionales suministrados por SAP, también puede crear sus propios grupos de funciones y módulos de función.

Cada grupo de funciones se conoce por un identificador de cuatro caracteres denominada *ID del grupo de funciones* . Si crea su propio grupo de funciones, debe elegir un ID de grupo de funciones que comienza con *Y* o *Z* .El ID debe ser exactamente de cuatro caracteres de longitud y no puede contener espacios en blanco ni caracteres especiales.

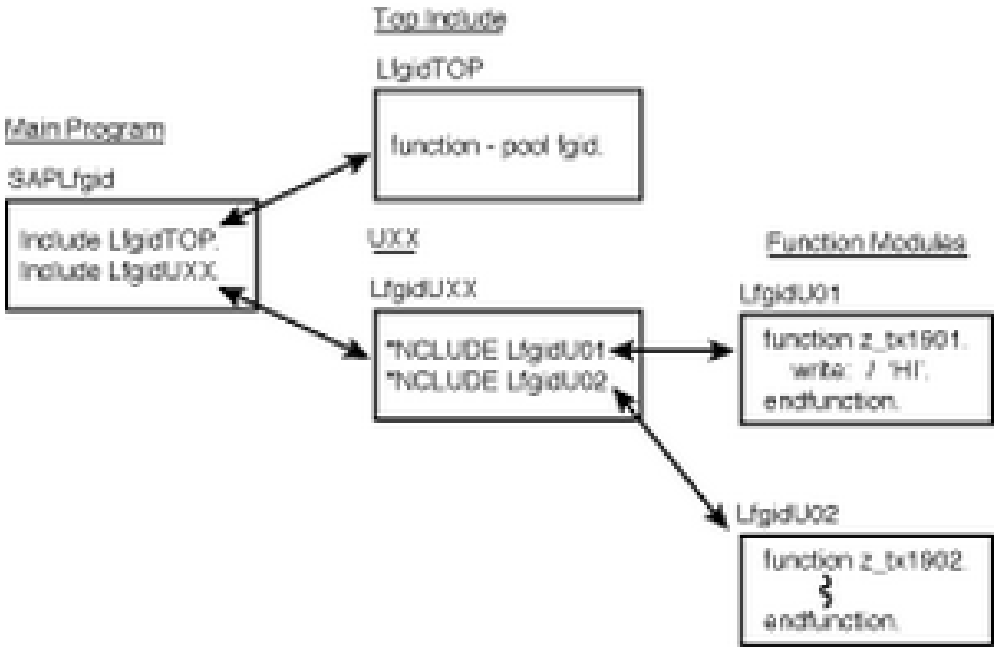
Para ilustrar los grupos de funciones, voy a describir brevemente el proceso de creación de un grupo de funciones y un módulo de función dentro de él. Vamos a suponer que aún no existen ni el grupo de funciones, ni módulo de función alguno dentro del mismo.

Se empieza por la creación de un módulo de funciones. Cuando se crea un módulo de función, el sistema primero le pedirá un ID del grupo de funciones. Este ID le indica al sistema dónde almacenar el módulo de función.

Cuando usted proporciona el ID, y si todavía no existe, el sistema crea:

- Un programa principal
- Una `include` denominado *top*
- Un `include` *UXX*
- Un `include` para cada módulo de función

En conjunto, estos componentes son conocidos como el grupo de funciones. Sus relaciones se ilustran en la figura:



El nombre del programa principal será SAPLFGID , donde FGID es su identificación de cuatro caracteres del grupo de funciones. El sistema coloca automáticamente dos declaraciones `include` en él:

- `include lFGIDtop`
- `include lFGIDUxx`

El primer `include` -el programa `lFGIDtop` - se conoce como el `include` top. En su interior se puede colocar las definiciones de datos globales. Estas son definiciones de los datos que son globales para todos los módulos de función dentro del grupo.

El segundo `include` - el programa `lFGIDUxx` - se conoce como el *UXX* . No se le permite modificar el UXX. El sistema colocará automáticamente una declaración `include` en el UXX para cada módulo de función que se crea en este grupo de funciones. Para el primer módulo de la función, la declaración `include lFGIDu01` se insertará en el UXX. Cuando se cree un segundo módulo de funciones en este grupo, el sistema añadirá una segunda declaración: `include lFGIDu02` . Cada vez que se agrega un nuevo módulo de función para el grupo, el

sistema añadirá una nueva declaración `include` a la UXX. El número del `include` será `01` para el primer módulo de función creado dentro del grupo, `02` para el segundo, `03` para el tercero, y así sucesivamente.

Dentro de cada `include` mencionado en el UXX está el código fuente de un módulo de función.

Acceso a la biblioteca de funciones

Los grupos de función se almacenan en un grupo de tablas dentro de la base de datos denominado la *biblioteca de funciones* . Al acceder a la biblioteca de funciones, también se puede acceder a las herramientas con las que se manipulan módulos y grupos de funciones. Para acceder a la biblioteca de funciones del *Workbench*, utilice el código de transacción es `SE37` .

Cada uno de los componentes de un grupo de funciones se puede acceder desde la biblioteca de funciones de la pantalla inicial de la SE37. Así, se puede acceder al código del programa principal, al include top y al código de cada include UXX (códigos de los módulos de función).

La activación de un módulo de funciones

Un módulo de función debe *activarse* antes de que pueda ser llamado por primera vez o si fue desactivado recientemente. Hay un botón Activar en la biblioteca de funciones de la pantalla inicial. Al pulsarlo se activa el módulo de función. Si la función ya está activada, este paso puede obviarse.

Cuando se crea por primera vez el módulo de función, la declaración `include` dentro del UXX está comentada con un *. La activación del módulo de función hace que el sistema elimine el comentario del `include`. El módulo de función está entonces disponible para ser llamado desde otros programas. Si cambia el código dentro del módulo de función, no es necesario que se active de nuevo (por otra parte, la reactivación no hará ningún daño, tampoco).

Si lo desea, puede desactivar a su vez un módulo funcional. Desde la pantalla inicial de la biblioteca de funciones, seleccione la ruta de menú:
Módulo> *Desactivar*. La elección de esta ruta de menús comenta la declaración `include` para el módulo de funciones dentro del UXX.

Todos los módulos de funciones que pertenecen a un grupo existen dentro del programa principal: `SAPL`*`FGID`* . Debido a esto, un error de sintaxis en uno cualquiera de estos módulos de función hará que el resto se convierta en

inoperante; la ejecución de cualquiera de ellos dará lugar a un error de sintaxis. Las funciones de activar y desactivar permiten trabajar en cualquier módulo de función de un grupo en forma única, sin afectar al resto. Puede

desactivar un módulo de función antes de trabajar en él, cambiar el código, y luego realizar una comprobación de sintaxis en ese único módulo antes de reactivarla. De esta manera, al resto de los módulos de función se le puede llamar, incluso mientras se está trabajando en una de ellas.

Definición de los datos dentro de un módulo de funciones

Las definiciones de datos dentro de los módulos de función son similares a los de subrutinas.

Dentro de un módulo de función, utilice la declaración `data` para definir las variables locales que se reinicializan cada vez que el módulo de función se llama. Utilice la declaración `statics` para definir las variables locales que se asignan la primera vez que el módulo de función se llama. El valor de una variable estática es recordado entre llamadas.

Defina los parámetros de la interfaz de módulo de función para crear las definiciones locales de las variables que se pasan al módulo de función y las que vuelven de él (véase la sección siguiente).

No se puede utilizar la declaración `local` dentro de un módulo de funciones. En su lugar, los parámetros de interfaz globalizados tienen el mismo propósito. Consulte la siguiente sección sobre la definición de los datos globales para aprender acerca de los parámetros de la interfaz local y global.

Definición de la interfaz de módulo de función

Para pasar parámetros a un módulo de función, se debe definir una *interfaz de módulo de función* . La interfaz de módulo de función es la descripción de los parámetros que se pasan y reciben desde el módulo de función. También se conoce simplemente como la *interfaz* . En el resto de este capítulo, me referiré a la interfaz de módulo de función simplemente como la interfaz.

Para definir los parámetros, hay que ir a una de las solapas de definición de los distintos tipos de parámetros, dentro de la SE37:

- Importación
- Exportación
- Importación/Exportación
- Tabla
- Excepciones

Los parámetros de importación son variables o cadenas de campos que contienen valores que se pasan al módulo de función desde el programa principal de llamada. Estos valores se originan fuera del módulo de función y se importan a él.

Los parámetros de exportación son variables o cadenas de campos que contienen valores devueltos por el módulo de función al programa que la invoca. Estos valores se originan dentro del módulo de función y se exportan fuera de ella.

Los parámetros de modificación, de importación/exportación o de I/O son variables o cadenas de campos que contienen valores que se pasan al módulo de función, son modificados por el código dentro del módulo de función, y luego regresan modificados al programa invocante. Estos valores se originan fuera del módulo de función. Ellos se pasan a él, cambian, y pasan de nuevo al programa principal.

Los parámetros de tabla son las tablas internas que se pasan al módulo de función como parámetros, cambian dentro de él, y regresan. Las tablas internas se deben definir en el programa de llamada.

Una *excepción* es un nombre para un error que se produce dentro de un módulo de función. Las excepciones se describen en detalle en la siguiente sección.

Pasar parámetros

Los métodos para pasar parámetros a los módulos de función son muy similares a aquellos para pasar parámetros a subrutinas externas.

Por defecto:

- los parámetros de importación y exportación se pasan por valor.
- los parámetros de modificación se pasan por valor y resultado.
- las tablas internas se pasan por referencia.

El uso de parámetros con tipo y sin tipo

Al igual que en las subrutinas, los parámetros de los módulos de función se pueden escribir con o sin tipo. El tipado funciona de la misma manera para los módulos de función como lo hace para las subrutinas. Los parámetros sin tipo adoptan todas sus características técnicas del paso de parámetros. Por ejemplo, si el parámetro pasado es del tipo `c` longitud 12 y el parámetro de módulo de función no tiene tipo, el parámetro de módulo de función se convierte en tipo `c` longitud 12.

Los parámetros con tipo de los módulos de función siguen las mismas reglas que los parámetros con tipo de las subrutinas. Hay dos formas de especificar los parámetros:

- Al especificar un tipo de datos ABAP / 4
- Al especificar el nombre de una estructura DDIC o un componente de una estructura DDIC

No puede utilizar ambos métodos al mismo tiempo para un solo parámetro. Debe utilizar uno u otro.

Usando el primer método, sólo tiene que especificar un tipo de datos ABAP / 4 . Usando el segundo método, introduzca el nombre de una estructura de diccionario de datos. Los parámetros se definen como si se hubiera usado la adición `like` en la declaración `form` en una subrutina externa.

Los parámetros también se pueden hacer opcionales y se les pueden dar valores por defecto. Si un parámetro de importación es opcional, no hay que nombrarlo en la declaración `call function` cuando se llama al módulo de función desde el programa principal. Los parámetros de exportación son siempre opcionales. Por eso, quiero decir que no es necesario codificar los parámetros de exportación en la declaración `call function`. Por ejemplo, si el módulo de función devuelve tres parámetros de exportación, `e1` , `e2` y `e3`, y sólo se desea `e2` , entonces sólo se debe incluir `e2` en la declaración `call function`. Simplemente omitir el resto.

Para dar a un parámetro de importación un valor predeterminado, también se permiten nombres de variables de la estructura *syst* (por ejemplo, se puede pasar un parámetro con valor por *default* igual a *sy-datum*, la fecha actual).

Todos los parámetros que se definen en la interfaz también aparecen en la parte superior del código fuente dentro de los comentarios especiales. Cada línea de comentario especial comienza con los caracteres `* "` . Cada vez que cambia la

interfaz del sistema se actualizan automáticamente estos comentarios especiales para reflejar los cambios.

PRECAUCIÓN: No cambie o borre estos comentarios, que son generados por el sistema. Si los cambia, el módulo de función podría no funcionar.

Por ejemplo, en la siguiente figura, que muestra el código de un módulo de función, se ve que se pasan dos parámetros a la función, por valor, uno de los cuales tiene un valor por defecto, y que se devuelve otro, también por valor:

```
1  function z_tx_div.
2  *"----------------------------------------------------------------------
3  *"*"Local interface:
4  *"       IMPORTING
5  *"             VALUE(P1) TYPE  I DEFAULT 1
6  *"             VALUE(P2) TYPE  I
7  *"       EXPORTING
8  *"             VALUE(P3) TYPE  P
9  *"----------------------------------------------------------------------
10 p3 = p1 / p2.
11 endfunction.
12
```

Llamando a los módulos de funciones

Para llamar a un módulo de función, se debe escribir la declaración `call function` . Veamos un ejemplo de un programa que realiza una simple llamada a un módulo de funciones:

```
1  report ztx0901.
2
3  write: / 'Antes'.
4  call function 'Z_TX_0901'.
5  write: / 'Después'.
```

El código anterior produce esta salida:

```
Antes
Hola desde Z_TX_0901
Después
```

El análisis del código anterior sería éste:

- La línea 4 transfiere el control al inicio del módulo de función `z_tx_0901` . En este ejemplo, suponemos que existe el módulo de función dentro de algún grupo de funciones.
- El código dentro del módulo de función se ejecuta (en este caso, el código sería: `write` 'Hola desde Z_TX_0901'.).
- La última línea del módulo de función devuelve el control a la declaración `call function` .
- El proceso continúa con la siguiente declaración después de `call function` .

Sintaxis para la declaración de llamada de la función (call function)

La siguiente es la sintaxis de la declaración `call function`:

```
call function 'F'
    [exporting   p1 = v1 ... ]
    [importing   p2 = v2 ... ]
    [changing    p3 = v3 ... ]
    [tables      p4 = it ... ]
    [exceptions  x1 = n [others = n]].
```

donde:

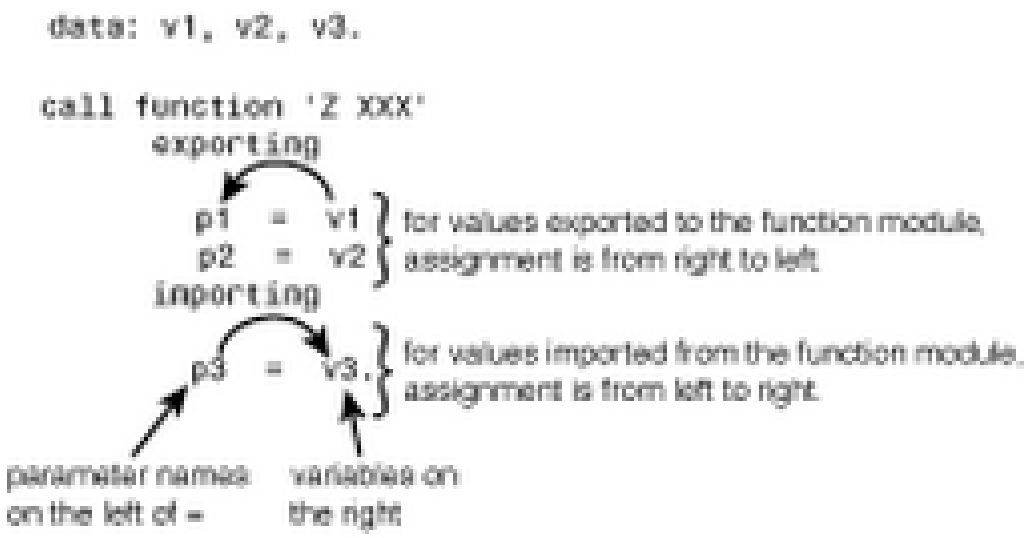
- `F` es el nombre del módulo de función.
- *p1* a *p4* son los nombres de los parámetros definidos en la interfaz del módulo de función.
- *v1* a *v3* son los nombres de campos o variables definidos en el programa de llamada.
- *it* es una tabla interna definida en el programa
- *n* es un entero o literal, no puede ser una variable.
- *x1* es un nombre de la excepción levantada en el módulo de función.

Los siguientes puntos se aplican:

- Todas las adiciones son opcionales.
- `call function` es una sola sentencia. No coloque puntos o comas después de los parámetros o nombres de excepción.
- El nombre del módulo de función debe estar codificado en mayúsculas. Si se codifica en minúsculas, la función no se encontrará y resultará un *short dump* (error en tiempo de ejecución).

Utilice la declaración `call function` para transferir el control a un módulo de función y especificar los parámetros.

La figura ilustra cómo se pasan y cómo son recibidos los parámetros desde el módulo de función:



En la declaración `call function`, las exportaciones y las importaciones son denominadas desde el punto de vista del programa. Los valores que son *exportados* por `call function` se importan en el módulo de función. Los valores exportados desde el módulo de función se *importan* en la declaración la `call function`.

A la izquierda del operador de asignación (=) hay una lista de los nombres de los parámetros definidos en la interfaz del módulo de función. A la derecha están las variables definidas dentro del programa de llamada. Las asignaciones después de `exporting` proceden de derecha a izquierda. Las asignaciones después de `importing` proceden de izquierda a derecha. Las asignaciones después de `changing` y `tables` son bidireccionales. Antes de la llamada, la asignación es de derecha a izquierda. En cambio, el valor de retorno se asigna de izquierda a derecha.

Vamos a utilizar la figura como ejemplo. Cuando se llama al módulo de función `Z_XXX` , el valor de la variable `v1` se asigna al parámetro `p1` y el valor de la variable `v2` se asigna a `p2` .Después, el control se transfiere al módulo de función. En cambio, al regresar de la función, el valor de `p3` se asigna a `v3` .

Ejecución de un módulo de función de ejemplo

En esta sección voy a ilustrar una llamada a un módulo de función de ejemplo `Z_TX_DIV` , que simplemente divide dos números y devuelve el resultado de la división.

El listado del programa 0902 llama a `Z_TX_DIV` y pasa parámetros:

```
1  report ztx0902.
2  parameters: op1 type i default 2,   "operando 1
3              op2 type i default 3.   "operando 2
4  data rslt type p decimals 2.        "resultado
5
6  call function 'Z_TX_DIV'
7       exporting
8            p1      = op1
9            p2      = op2
10      importing
11           p3      = rslt.
12
13 write: / op1, '/', op2, '=', rslt.
```

El código de la función es el siguiente:

```
1  function Z_TX_DIV.
2  *"----------------------------------------------------------
3  *"*"Local interface:
4  *"       IMPORTING
5  *"             VALUE(P1) TYPE  I DEFAULT 1
6  *"             VALUE(P2) TYPE  I
7  *"       EXPORTING
8  *"             VALUE(P3) TYPE  P
9  *"----------------------------------------------------------
10 p3 = p1 / p2.
11 endfunction.
```

El código de la función se llama desde la línea 6 del ztx0905.

El código del listado ztx0902 produce esta salida:

```
2/3 = 0,67
```

- En el listado ztx0902, las líneas 2 y 3 definen dos variables, `OP1` y `OP2` . Debido a que éstas se definen mediante el declaración `parameters`, el usuario debe ingresar sus valores para estas variables cuando se ejecuta el programa.
- La línea 4 define una variable llamada `rslt` del tipo `p` con dos cifras decimales.

- La línea 6 transfiere el control al módulo funcional `Z_TX_DIV` . El valor de la variable `OP1` se asigna al parámetro `P1` y el valor de `OP2` se asigna al parámetro `P2` . Los tipos de datos de `OP1` y `OP2` deben coincidir con los tipos de datos de `p1` y `p2` . `OP1` y `OP2` son definidos como números enteros, por lo tanto, coinciden con los tipos de datos. `P3` es de tipo `p` , es un parámetro parcialmente tipado. Obtiene su longitud y el número de decimales de `rslt` . Los valores de `OP1` y `OP2` son transferidos al módulo de función y el control a la línea 1 del mismo.
- En el módulo de función, la línea 10 realiza una división simple.
- La función devuelve el control a la línea 6 del programa. El valor de `P3` se asigna a la variable `RSLT` .
- La línea 13 escribe el resultado de la división.

TIP
Pulse el botón PATTERN en el Editor ABAP / 4, pantalla de edición de programas, para insertar automáticamente la declaración `call function` en el código. Se le preguntará por el nombre del módulo de función y de forma automática se introducirán los códigos la parte izquierda de todas las asignaciones de parámetros de interfaz. No deje en blanco agregados `importing` o `exporting`, pues luego obtendrá un error al compilar.

Creación de un módulo de función

Utilice el procedimiento estándar, a partir de la transacción SE37, para crear un módulo de función:

- Escriba el nombre de su módulo de funciones en el campo Módulo de funciones.
- El nombre debe comenzar con *Y_* o *Z_* .
- Pulse el botón Crear.
- Escriba el nombre de un grupo de funciones en el campo Grupo de funciones. El nombre del grupo de funciones debe ser de cuatro caracteres de longitud y debe comenzar con *Yo Z (*si no está creado, deberá crearlo ahora, como se verá enseguida).
- Escriba una **S** en el campo de aplicación. Este campo se utiliza para indicar qué área de función utiliza el módulo de función. Nuestra función no es utilizada por cualquier área funcional, es simplemente un ejemplo, por lo que cualquier opción sirve. (S indica que el módulo de función contiene funcionalidad necesaria para tareas de Basis.).
- Escriba una descripción del módulo de funciones en el campo de texto. El contenido de este campo se ve cuando se muestra una lista de los módulos de función.
- Pulse el botón Guardar en la barra de herramientas de la aplicación.

- Si el grupo función no existe, un mensaje emergente le informa de este hecho y le preguntará si desea crearlo. Pulse el botón Sí para crear el grupo de funciones. Aparecerá el cuadro de diálogo *Crear grupo de funciones*.
- Escriba una descripción en el campo de texto y pulse el botón Guardar. Aparece la pantalla de entrada de catálogo de objetos Crear.
- Pulse el botón Objeto Local.
- Vaya a la solapa *Código fuente* en la SE37. Se muestra la pantalla de edición del módulo de funciones (véase la figura anterior con el código de función).
- Escriba el código fuente de su módulo de función. No cambie las líneas de comentarios generados por el sistema bajo ninguna circunstancia! Su módulo de función podría no funcionar si lo hace.
- Pulse el botón Guardar en la barra de herramientas de la aplicación. El mensaje Programa xxxxx guardado aparece en la barra de estado situada en la parte inferior de la ventana.
- Pulse el botón Atrás de la barra de herramientas de la aplicación. Regresará a la pantalla inicial.
- Si desea definir los parámetros de importación o de exportación, hágalo. Escriba los nombres de los parámetros en la primera columna y escriba las otras características deseadas para cada parámetro. Cuando haya terminado, pulse el botón Guardar y luego en el botón Atrás.
- Por último, para activar su módulo de función, presione el botón Activar en la barra de herramientas Aplicación de la pantalla inicial.
- Dentro de su programa, asegúrese de codificar el nombre de módulo de función en mayúsculas. Si no lo hace, obtendrá un *short dump* debido a que el sistema no será capaz de encontrar el módulo de función.

En resumen, hemos visto hasta aquí que:

- Los módulos de función son unidades de modularización con nombres únicos que se pueden llamar desde cualquier programa en todo el sistema R / 3. Ellos tienen código que puede ser utilizado por más de un programa.
- Un grupo de funciones es una colección de programas y tiene una estructura predefinida. Los nombres de todos los programas dentro del grupo contienen los cuatro caracteres de identificación del grupo de funciones.
- La interfaz de un módulo de función contiene las definiciones los parámetros de importación, exportación, y de I/O, así como la documentación de las excepciones planteadas dentro del módulo de función.
- Pueden ser especificados parámetros con tipo y sin tipo y pueden ser pasados por referencia, por valor o por su valor y resultado.

Veamos algunos *tips* generales para el uso de funciones:

- Pase por referencia los parámetros de importación, es más eficiente. Tenga cuidado al usarlo con parámetros de exportación.
- **NO** cambiar los comentarios generados por el sistema en la parte superior del código fuente del módulo de función.

- Guarde los datos de prueba para que pueda volver a probar el módulo de función después de las modificaciones utilizando los mismos datos.
- **NO** importar valores que no necesita a un módulo funcional. Simplemente omita las importaciones que no sean necesarias desde la declaración `call function`.
- Documente todas las excepciones y parámetros con el componente de documentación del módulo de función.
- **NO** utilice `stop`, `exit`, o `check` dentro de un módulo de función. (`check` y `exit` están bien si están dentro de un bucle en el interior del módulo de función.)

Definir datos globales para los módulos de función

Dos tipos de *datos globales* se pueden definir para un módulo de función:

- Las definiciones de datos colocados en la parte superior `son` accesibles desde todos los módulos de funciones y subrutinas dentro del grupo. Este tipo de datos globales es persistente a través de llamadas de módulo de función.
- Las definiciones de datos dentro de la interfaz sólo se conocen dentro del módulo de función. Si la interfaz se define como una *interfaz global* , las definiciones de parámetros también son conocidos dentro de todas las subrutinas que llama al módulo de funciones. Este tipo de datos global **no** es persistente a través de llamadas de módulo de función.

Los datos globales definidos en el `include` *top* es accesible desde todos los módulos de función dentro del grupo. Es análogo a las definiciones de los datos globales de la parte superior de un programa ABAP / 4.

Los parámetros definidos dentro de una *interfaz global* son accesibles desde todas las subrutinas llamadas desde el módulo de función. Es análogo a las variables definidas dentro de una subrutina mediante la sentencia `local`.

Los parámetros definidos dentro de un *interfaz local* son inaccesibles desde las subrutinas llamadas desde el módulo de función. Es análogo a las variables definidas *dentro* de una subrutina usando la declaración `data`. De forma predeterminada, las interfaces son locales.

Los datos globales definidos en el `include` *top* se asignan cuando un programa hace su primera llamada a cualquier módulo de función de un grupo. Sigue estando asignado siempre y cuando el programa de llamada permanezca activa. Cada llamada posterior a un módulo de funciones dentro del mismo grupo ve los valores previos dentro del área de datos global. Los valores en el área de datos global persisten hasta que finalice el programa de llamada.

Los datos globales definidos mediante el interfaz global se reinician cada vez que el módulo de función es llamado.

Las siguientes líneas de código ilustran la persistencia de los datos globales definidos en el `include` *top*.

```
1 report ztx0903.
 2 parameters parm_in(10) default 'XYZ' obligatory.
 3 data f1(10) value 'INIT'.
 4
 5 perform: call_fm12,
 6          call_fm13.
 7 write: / 'f1 =', f1.
 8
 9 *
10 form call_fm12.
11   call function 'Z_TX_0102'
12        exporting
13             p_in    = parm_in
14        exceptions
15             others  = 1.
16   endform.
17
18 *
19 form call_fm13.
20   call function 'Z_TX_0103'
21        importing
22             p_out   = f1
23        exceptions
24             others  = 1.
25   endform.
```

Este es el código fuente del módulo de la primera función llamada en el listado anterior:

```
1 function z_tx_0102.
2 * "----------------------------------------- -------------
3 * "*" Interfaz local:
4 * "IMPORTACIÓN
5 * "VALUE (P_IN)
6 * "----------------------------------------- -------------
7 g1 = P_IN.
8 endfunction.
```

Y este es el código fuente del módulo de la segunda función llamada en el listado anterior:

```
1 function z_tx_0103.
2 * "----------------------------------------------- -------------
3 * "*" Interfaz local:
4 * "EXPORTACIÓN
5 * "VALUE (P_OUT)
6 * "----------------------------------------------- -------------
7 p_out = g1.
8 endfunction.
```

Y este es el `include` *top:*

```
1 function-pool ztxb.
2 data G1(10).
```

Esta es la salida:

```
f1 = XYZ
```

Este es el análisis:

- En el primer listado, de la línea 5 se transfiere el control a la línea 10 y la línea 11 llama al módulo de función `z_tx_0102` , que existe en el grupo de funciones `ztxb` . El valor de `parm_in` se pasa a `P_IN` . Se transfiere el control a la línea 1 del segundo listado.
- En el listado, en la línea 7 se asigna el valor de `P_IN` a la variable global `g1` . `g1` se define en la línea 2 del `include` *top*. Eso hace que sea visible para todos los módulos de función y el valor es persistente entre llamadas los módulos de función de ese grupo. El control vuelve a la línea 11 del primer listado.
- En ese listado, la línea 16 transfiere el control a la línea 5, luego a la línea 6, y luego a la línea 19. La línea 20 llama al componente funcional `z_tx_2003` , que también existe en el grupo de funciones `ztxb` . Se transfiere el control a la línea 1 del tercer listado.
- En el tercer listado, en la línea 7 se asigna el valor de la variable global `g1` a `p_out` . `g1` todavía contiene el valor que le fue asignado por el módulo de función anterior. Se devuelve el control al programa de llamada y se escribe el valor de salida.

Pasando una tabla interna a un módulo de función

Pasar una tabla interna de un módulo de función sigue las mismas reglas que la de pasar una tabla interna a una subrutina. Hay dos métodos:

- A través de la parte `tables` e la interfaz. Esto es equivalente a pasar a través de la adición `tables` de la declaración `perform`.

- A través de la parte `importing` o `changing` de la interfaz. La primera es equivalente a `using` en la declaración `perform` y lo último es equivalente a `changing` en `perform` .

Para devolver una tabla interna que se origina desde el interior de un módulo de función, utilice uno de los siguientes:

- La parte `tables` de la interfaz
- La parte `exporting/importing/changing` de la interfaz

Pasando una tabla interna vía *tables*

Si utiliza la parte `tables` e la interfaz, se puede pasar la tabla interna junto con la línea de cabecera. Si no tiene una línea de cabecera, adquirirá automáticamente uno dentro del módulo de función. Los parámetros del tipo `tables` siempre se pasan por referencia.

Si la tabla interna es de un tipo estructurado, puede escribirse opcionalmente el nombre de una estructura DDIC. Si no se especifica una estructura DDIC aquí, usted no será capaz de acceder a cualquiera de los componentes de la tabla interna en el módulo de función. Ello provocará un error de sintaxis. Si la tabla interna no es un tipo estructurado, se puede introducir un tipo de datos ABAP / 4 en su lugar.

Pasando una tabla interna vía *importing*, *exporting* o *changing*

Con este método se puede pasar sólo el cuerpo de una tabla interna. Sin embargo, estas partes de la interfaz le permiten elegir si desea pasar el cuerpo de la tabla interna por valor (por defecto) o por referencia. Al utilizar este método, debe especificar `table` en la columna de tipo de referencia.

Los siguientes listados ilustran las formas de pasar las tablas internas de un módulo funcional, comenzando con el código del programa:

```
 1 report ztx0904.
 2 tables spfli.
 3 data: it_a like spfli occurs 3,   "doesn't have a header line
 4       it_b like spfli occurs 3.   "doesn't have a header line
 5 select * up to 3 rows from spfli into table it_a.
 6
 7 call function 'Z_TX_0906'
 8      exporting
 9           it1     = it_a[]
10      importing
11           it2     = it_b[]
12      tables
```

```
13           it4     = it_a
14      changing
15           it3     = it_a[]
16      exceptions
17           others  = 1.
18
19 write: / 'AFTER CALL',
20        / 'IT_A:'.
21 loop at it_a into spfli.
22     write / spfli-carrid.
23     endloop.
24 uline.
25 write / 'IT_B:'.
26 loop at it_b into spfli.
27     write / spfli-carrid.
28     endloop.
```

Y continuando con el código de la función referenciada:

```
1  function z_tx_0906.
2  *"----------------------------------------------------------
3  *"*"Local interface:
4  *"       IMPORTING
5  *"             VALUE(IT1) TYPE  TABLE
6  *"       EXPORTING
7  *"             VALUE(IT2) TYPE  TABLE
8  *"       TABLES
9  *"             IT4 STRUCTURE  LFA1
10 *"       CHANGING
11 *"             VALUE(IT3) TYPE  TABLE
12 *"----------------------------------------------------------
13 data wa like lfa1.
14 write / 'IT1:'.
15 loop at it1 into wa.
16     write / wa-carrid.
17     endloop.
18 loop at it3 into wa.
19     wa-carrid = sy-tabix.
20     modify it3 from wa.
21     endloop.
22 uline.
23 write: / 'IT4:'.
24 loop at it4.
25     write / it4-carrid.
26     endloop.
27 uline.
28 it2[] = it1[].
29 endfunction.
```


Definición de subrutinas en un grupo de funciones

Usted puede llamar a subrutinas internas y externas desde dentro de los módulos de función. Las llamadas a subrutinas externas son las mismas para los módulos de función, que las que se usan para los informes. Las subrutinas internas deben ser definidas dentro de un `include` especial: el *F01*.

El `include` *F01* se crea en el grupo de funciones al editar el programa principal del grupo de funciones, con la inserción de la declaración `include L`*`FGID`*`F01` . A continuación, haciendo doble clic sobre el nombre del `include` se creará automáticamente. Dentro del mismo, se puede poner subrutinas accesibles por todos los módulos de función dentro del grupo. Hay, sin embargo, una manera más fácil. La forma más sencilla de definir el *F01* es codificar su llamada a la rutina y hacer doble clic en el nombre de la subrutina en la declaración `perform` .

Liberación de un módulo de función

El comando `release function` añade un nivel de protección a la interfaz de un componente funcional al protegerla de modificación. Si se ha completado las pruebas del módulo de función, es posible que se desee *liberarlo* e indicar que es seguro para otros desarrolladores utilizar en su código. Al hacerlo, se está esencialmente asegurando de no cambiar los parámetros de interfaz existentes en ninguna forma que pueda causar problemas a todos los programas que llamarán a ese módulo de función. Se está prometiendo estabilidad de la interfaz.

Por ejemplo, después de la liberación de un módulo de función, no debe agregarse, cambiar, o eliminar un parámetro. Si es así, las llamadas existentes a ese módulo de función pueden fallar o funcionar de forma diferente.

Después de la liberación, todavía puede cambiarse la interfaz, pero hay que dar un paso más para hacerlo.

Prueba de un módulo de función

Si usted desea probar un módulo de funciones sin necesidad de escribir un programa ABAP / 4, puede hacerlo utilizando el entorno de prueba dentro de la biblioteca de funciones.

Encontrar módulos de función existentes

Como se dijo antes, SAP suministra más de 5.000 grupos de funciones pre-existentes que contienen más de 30.000 módulos de función. A menudo, la funcionalidad que necesita ya está cubierta por estos módulos de función suministrados por SAP, sólo se tiene que encontrar la correcta. Para encontrar un módulo de función existente, utilice el sistema de información del repositorio (Repository Information System), mediante la transacción SE80.

Una vez encontrado el módulo, se puede corregir errores de sintaxis o modificar parámetros, se puede determinar a qué grupo pertenece, se pueden explorar los componentes de grupo (en particular, el módulo previamente encontrado),etc.

Establecer el valor de *sy-SUBRC* en el retorno

Normalmente, después de regresar de un módulo de función, el sistema ajusta automáticamente el valor de `sy-SUBRC` a cero. Utilice uno de los siguientes dos comandos para configurar `sy-SUBRC` a un valor distinto de cero:

- `raise`
- `message ... raising`

Uso de la instrucción *raise*

Utilice la declaración `raise` para salir del módulo de función y establecer el valor de `sy-SUBRC` a su regreso.

Sintaxis para la declaración de *raise*

La siguiente es la sintaxis para `raise`:

```
raise XName.
```

donde:

- *`XName`* es el nombre de la excepción a ser levantada.

Los siguientes puntos se aplican:

- *`XName`* puede ser cualquier nombre que se invente. No tiene que ser definido previamente en ningún lugar. Puede tener hasta 30 caracteres de longitud. Todos los caracteres están permitidos a excepción de `" '. , y : .`
- No encierre *`XName`* entre comillas.
- *`XName`* no puede ser una variable.

Cuando se ejecuta la sentencia `raise`, el control vuelve inmediatamente a la declaración `call function` y se asigna un valor de `sy-SUBRC` basado en las excepciones que se han enumerado allí. Los valores asignados a los parámetros de exportación no se copian de nuevo al programa de llamada.

Los siguientes listados ilustran cómo sucede esto:

```
1 report ztx0907.
 2 parameters parm_in default 'A'.
 3 data vout(4) value 'INIT'.
 4 call function 'Z_TX_0908'
 5     exporting
 6         exname = parm_in
 7     importing
 8         pout   = vout
 9     exceptions
10         error_a = 1
11         error_b = 4
12         error_c = 4
13         others  = 99.
14 write: / 'sy-subrc =', sy-subrc,
15        / 'vout =', vout.
```

El código de la función es el siguiente:

```
1 function z_tx_0908.
 2 *"----------------------------------------------------------------------
 3 *"*"Local interface:
 4 *"       IMPORTING
 5 *"             VALUE(EXNAME)
 6 *"       EXPORTING
 7 *"             VALUE(POUT)
 8 *"       EXCEPTIONS
 9 *"              ERROR_A
10 *"              ERROR_B
11 *"              ERROR_C
12 *"              ERROR_X
13 *"----------------------------------------------------------------------
14 pout = 'XXX'.
15 case exname.
16   when 'A'. raise error_a.
17   when 'B'. raise error_b.
18   when 'C'. raise error_c.
19   when 'X'. raise error_x.
20   endcase.
21 endfunction
```

La salida del listado es ésta:

```
sy-subrc =     1
vout = INIT
```

El análisis es éste:

- En el listado del programa, la línea 4 toma el valor de `parm_in` y lo pasa al módulo de funciones `z_tx_0907` . Transfiere el control a la línea 1 del segundo listado.
- En el segundo listado, la línea 14 asigna un valor al parámetro `pout`. Este parámetro se pasa por valor, por lo que el cambio es sólo en la definición local de `pout` . El original todavía no ha sido modificado.
- En el listado de la función, en la línea 15 se examina el valor que se pasa a través del parámetro `exname` . El valor es `B` , por lo que la línea 17 - `raise error_b` -se ejecuta. Transfiere el control a la línea 9 del listado del programa. Debido a que el la declaración `raise` ha sido ejecutada, el valor de `pout` no se copia de nuevo al programa de llamada, por lo que el valor de `vout` no se modificará.
- En el primer listado, el sistema escanea las líneas 10 a 13 hasta que se encuentra una coincidencia para la excepción nombrada por la recién ejecutada sentencia `raise` . En este caso, está en busca de `error_b` .La línea 11 coincide.
- En la línea 11, el valor en el lado derecho del operador igual se asigna a `sy-SUBRC` . Después, el control se transfiere a la línea 20.

El nombre especial `others` en la línea 13 del listado del programa coincidirá con todas las excepciones que no se citan explícitamente en la adición `exceptions`. Por ejemplo, si la línea 19 del listado de la función fuera ejecutada, la excepción `error_x` sería levantada. Esta excepción no se nombra en el listado del programa, por lo que va a coincidir `others` y el valor de `sy-SUBRC` se establecerá en `99` . Puede codificarse el número de códigos de retorno de excepción que desee.

PRECAUCIÓN
Si no codificara `others` y se produce una excepción dentro del módulo de función que no se nombra en `exceptions`, el programa será abortado y resultará un *short dump* que tiene el error de ejecución *RAISE_EXCEPTION* .

Si un parámetro de exportación se pasa por valor, después de un `raise` su valor se mantiene sin cambios en el programa de llamada, incluso si un valor se asigna dentro del módulo de función antes de que `raise` sea ejecutado. Si el parámetro se pasa por referencia, se puede cambiar en el programa que llama Este efecto se muestra en el listado ztx0907. El valor de `pout` se cambia en el módulo de función y, si se ejecuta la sentencia `raise`, el valor modificado no se copia de nuevo en el programa de llamada.

NOTA
`check` , `exit` , y `stop` tienen el mismo efecto con los módulos de función como lo hacen en las subrutinas externas. Sin embargo, no deben ser usados dentro de los módulos de función. En cambio, el la declaración `raise` se debe utilizar, ya que permite al programa que llama establecer el valor de `sy` - `SUBRC` a su regreso.

Uso de la sentencia message ... raising

La declaración `message ... raising` tiene dos modos de funcionamiento:

- Si la excepción nombrada después de `raising` no se maneja por la declaración `call function` y `others` no está codificado, se emitirá un mensaje al usuario.
- Si la excepción nombrada después `raising` está a cargo de la declaración `call function`, no se emite un mensaje para el usuario. En cambio, el control vuelve a la declaración `call function` y la excepción se maneja de la misma manera que para `raise`.

Sintaxis para la declaración *message ... raising*

La siguiente es la sintaxis para la declaración `message ... raising`

```
message tnnn(cc) [with v1 v2 ...] raising xname
```

donde:

- *`t`* es el tipo de mensaje (`e` , `w` , `i` , `s` , `a` , o `x`).
- *`nnn`* es el número de mensaje.
- *`(Cc)`* es la clase de mensaje.
- *`v1`* y *`v2`* son valores que se insertan en el texto del mensaje.
- *`XName`* es el nombre de la excepción a ser levantada.

Los siguientes puntos se aplican:

- *`XName`* es un nombre de excepción como se describe para `raise`.
- Si no se especifica la clase de mensaje aquí, debe ser especificada en la declaración `función-pool` en el `include` *top* para el grupo de funciones a través de la adición `message-id` .
- Los siguientes variables `sy` son fijadas: `sy-msgid` , `sy-msgty` , `sy-msgno` y `sy-msgv1` a `sy-msgv4` . Estos pueden ser examinados en el programa de llamada (después del regreso).

Cuando se ejecuta la sentencia `message...raising`, el flujo de control depende del tipo de mensaje y si la condición es manejada por el programa que llama (especificado en la lista de excepciones en el programa de llamada).

- Si la condición es manejada por el programa que llama, el control vuelve al programa que llama. Los valores de las variables `sy-msg` son fijadas. No se devolverán los valores de las exportaciones pasados por valor.
- Si la condición no está a cargo del programa que llama, y para tipos de mensajes `e` , `w` , y `a` , se sale del módulo de función y el programa de llamada se termina inmediatamente. Se muestra el mensaje y la lista de salida se ve en blanco. Cuando el usuario presiona la tecla *Intro*, se sale de la lista en blanco y el usuario vuelve a la pantalla desde la que se invoca el programa.
- Si la condición no está a cargo del programa invocante, y para tipo de mensaje `i` , se muestra el mensaje al usuario en un cuadro de diálogo. Cuando el usuario presiona *Enter*, el control vuelve al módulo de función en la instrucción que sigue a `message`. El módulo de función sigue con el procesamiento y, al llegar a la declaración `endfunction` , el control vuelve al programa de llamada. Los valores de las variables `msg-sy` se ajustan y se devuelven los valores de las exportaciones pasados por valor.
- Si la condición no está a cargo del programa que llama, y para tipo de mensaje `s` , el mensaje se almacena en un área del sistema. El control continúa en el módulo de función en la instrucción que sigue a `message`. El módulo de función sigue con el procesamiento y, al llegar a la declaración `endfunction` , el control vuelve al programa de llamada. Los valores de las variables `msg-sy` se ajustan y se devuelven los valores de las exportaciones pasados por valor. Cuando aparezca la lista, aparecerá el mensaje en la barra de estado situada en la parte inferior de la lista.
- Si la condición no está a cargo del programa que llama, y para tipo de mensaje `x` , se sale del módulo de función y el programa de llamada se termina inmediatamente. Un *short dump* se genera y se muestra eso al usuario.

Los siguientes listados ilustran el efecto de la sentencia `message ... raising`:

```
 1 report ztx0909.
 2 parameters: p_etype default 'E', "ingrese tipos de mje E W I S A X
 3             p_handle default 'Y'.   "si es Y, maneja la excepción
 4 data vout(4) value 'INIT'.
 5
 6 if p_handle = 'Y'.
 7     perform call_and_handle_exception.
 8 else.
 9     perform call_and_dont_handle_exception.
10     endif.
11 write: / 'sy-subrc =', sy-subrc,
12        / 'vout =', vout,
13        / 'sy-msgty', sy-msgty,
14        / 'sy-msgid', sy-msgid,
15        / 'sy-msgno', sy-msgno,
16        / 'sy-msgv1', sy-msgv1.
17
18*_____________________________________________________________________
19 form call_and_handle_exception.
20   call function 'Z_TX_0910'
21      exporting
22           msgtype = p_etype
23      importing
24           pout    = vout
25      exceptions
26           error_e = 1
27           error_w = 2
28           error_i = 3
29           error_s = 4
30           error_a = 5
31           error_x = 6
32           others  = 7.
33   endform.
34
35*_____________________________________________________________________
36 form call_and_dont_handle_exception.
37   call function 'Z_TX_0910'
38      exporting
39           msgtype = p_etype
40      importing
41           pout    = vout.
42   endform.
```

Este es el código de la función:

```
 1 function z_tx_0910.
 2 *"----------------------------------------------------------------------
 3 *"*"Local interface:
 4 *"       IMPORTING
 5 *"             VALUE(MSGTYPE)
 6 *"       EXPORTING
 7 *"             VALUE(POUT)
 8 *"       EXCEPTIONS
 9 *"              ERROR_E
10 *"              ERROR_W
11 *"              ERROR_I
12 *"              ERROR_S
13 *"              ERROR_A
14 *"              ERROR_X
15 *"----------------------------------------------------------------------
16 pout = 'XXX'.
17 case msgtype.
18   when 'E'. message e991(zz) with 'Error E' raising error_e.
19   when 'W'. message e992(zz) with 'Error W' raising error_w.
20   when 'I'. message e993(zz) with 'Error I' raising error_i.
21   when 'S'. message e994(zz) with 'Error S' raising error_s.
22   when 'A'. message e995(zz) with 'Error A' raising error_a.
23   when 'X'. message e996(zz) with 'Error X' raising error_x.
24   endcase.
25 endfunction.
```

Ésta es la salida obtenida:

```
sy-subrc =     1
vout = INIT
sy-msgty E
sy-msgid ZZ
sy-msgno 991
sy-msgv1 Error E
```


Este es el análisis:

- En el listado del programa, al utilizar los valores predeterminados para el parámetro, la línea 6 es verdadera, por lo que el control se transfiere a la subrutina `call_and_handle_exception` .
- La línea 20 transfiere el control a la línea 1 del listado de la función.
- En el ese listado, la línea 16 asigna un valor al parámetro local `pout` . Este valor se define como pasaje por valor, por lo que el valor de `vout` en el programa que llama no se cambia.
- La línea 18 del enunciado `case` es verdadera y se ejecuta la sentencia `message ... raising`. Esto desencadena la excepción `error_e` .
- La excepción `error_e` es manejada por el programa de llamada, por lo que el mensaje no se muestra al usuario. El control vuelve a la línea 25 del listado del programa. El valor de `vout` no cambia; son asignados valores a las variables `sy-msg` .
- En el listado del programa, la línea 26 asigna `1` a `sy-SUBRC` .
- La línea 33 devuelve el control a la línea 11 y los valores se escriben.

Definición de excepciones en la interfaz

Cada vez que usted levanta una excepción en un módulo de función, se debe documentar ese nombre en la interfaz del módulo de función, en la solapa correspondiente. Los nombres de excepción aparecerán entonces automáticamente en el módulo de función, en la parte superior del código fuente, dentro de los comentarios generados por el sistema.