ABAPZombie Guide to ABAP – Parte 19 – CALL FUNCTION

21/03/2011 10:21

CALL FUNCTION, comando usado em 10 de cada 6 programas em ABAP.

O que ele faz? É só colocar CALL FUNCTION no Google Translation (en para pt) que vc descobre! 🙄

Bom, basicamente o comando serve para efetuar a chamada de qualquer função presente na transação SE37, como segue:

```
REPORT  zaz_call_function.

* Chamada Simples de Função
CALL FUNCTION 'POPUP_TO_INFORM'
  EXPORTING
    titel         = 'Zumbis Estão Atacando!'
    txt1          = 'Cuidado com o Ataque de Zumbis.'
    txt2          = 'Ele irá ocorrer hoje, às 18h.'
          .
```

Você, **tem** que listar os parâmetros obrigatórios que estiverem na descrição da função na SE37 (importing/exporting/tables) quando usar este comando.

Uma coisa que não é obrigatória na chamada da função **mas deve ser usada** são as EXCEPTIONS, que correspondem ao tratamento dos diversos erros que a função pode retornar (usando o comando RAISE EXCEPTION).

Caso a função retorne um erro, o código da EXCEPTION será transportado para a variável de sistema SY-SUBRC, e você pode tratar o erro adequadamente.

Importante: Caso você não liste as EXCEPTIONs no programa que está dando o CALL FUNCTION, e a função disparar uma exceção o seu programa irá *"dumpar"*. Como você não gosta de dumps, use sempre as exceptions 😉

```
REPORT  zaz_call_function.

DATA: t_dados TYPE TABLE OF string.

* Chamada da Função listando os Exeception e tratando os
* erros com as mensagens padrão do sistema
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    filename                = 'C:teste.txt'
  TABLES
    data_tab                = t_dados
  EXCEPTIONS
    file_open_error         = 1
    file_read_error         = 2
    no_batch                = 3
    gui_refuse_filetransfer = 4
    invalid_type            = 5
    no_authority            = 6
    unknown_error           = 7
    bad_data_format         = 8
    header_not_allowed      = 9
    separator_not_allowed   = 10
    header_too_long         = 11
    unknown_dp_error        = 12
    access_denied           = 13
    dp_out_of_memory        = 14
```

```
    disk_full              = 15
    dp_timeout             = 16
    OTHERS                 = 17.
IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
          WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.
```

O comando CALL FUNCTION também possue outras particularidades, vejamos:

RFCs: Você pode usar o comando CALL FUNCTION para chamar funções de outros sistemas SAP, desde que haja uma conexão configurada entre os dois sistemas e função for do tipo "RFC" (configurável na primeira aba da SE37) no sistema destino. Chamamos essa "conexão" de **DESTINATION**, e é através dele que o SAP sabe onde deve conectar-se para executar a função. Exemplo:

```
REPORT  zaz_call_function.

* Declaração de Variáveis
DATA: t_dados TYPE TABLE OF string,
      v_dest  TYPE char10.

* Nome do DESTINATION (criado pela transação SM59)
v_dest = 'QAS_200'.

* Chamada de RFC
* Neste caso, a função abaixo será executada no ambiente
* QAS_200. A execução é síncrona, ou seja, o SAP irá
* esperar o retorno dos dados para continuar a execução
* do programa.
CALL FUNCTION 'Z_RFC_ZUMBI'
  DESTINATION v_dest
  TABLES
    dados_selecionados    = t_dados
  EXCEPTIONS
    communication_failure = 1 "Exception de RFC
    system_failure        = 2. "Exception de RFC
```

No caso acima, a função foi executada no sistema caracterizado pelo destinado V_DEST, e não no sistema onde o programa está rodando. **É extremamente importante utilizar os EXCPETIONs especiais para chamadas de funções remotas** (Exceptions de RFC alí em cima).

UPDATE TASK: A chamada em UPDATE TASK de uma função faz com que a função não seja executada automaticamente, mas sim guardada na memória para ser executada dentro do work process de atualização do sistema. Esse work process é disparado quando um programa executa o comando **COMMIT WORK**. Se você chamar mais de uma função em UPDATE TASK num mesmo programa, alguns parâmetros que podem ser definidos na SE37 vão controlar a sequência de execução da pilha.
Exemplo:

```
* Peguei o caso de uma transação standard, é mais fácil de explicar
* O programa aqui é o MJ1B1I01

      CALL FUNCTION 'J_1B_NF_DOCUMENT_PRINT' IN UPDATE TASK
           EXPORTING
                DOC_NUMBER         = DOCNUM
          EXCEPTIONS
               DOCUMENT_NOT_FOUND = 6
               MESSAGING_PROBLEM  = 7
               OTHERS             = 9.

* Esse programa faz parte da transação J1B1N, criação de Nota Fiscal.
* O sistema faz a chamada da função que imprime a nota fiscal antes
```

```
* da nota ser criada, mas usa o UPDATE TASK, portanto
* a função aqui só vai ser disparada depois que o COMMIT WORK
* for executado, o que, no cenário da J1B1N indica que o doc.
* já foi gerado! Malandro, não é não?
```

STARTING NEW TASK: velho conhecido da galera adepta do paralelismo. Na verdade, a idéia aqui é fazer a execução de uma função RFC de modo assíncrono, ou seja, o programa chama uma função de outro sistema e não espera resposta. Porém, muita gente usa esse comando por aí para chamar uma função de modo assíncrono **dentro do próprio sistema**, abrindo assim várias tasks que rodam em paralelo. Esse é o esquema mágico paralelismo, que sempre assombra a vida de quem dá manutenção em programas malucos. Exemplo:

```
*&---------------------------------------------------------------------*
*& Report  ZMRC_TEST1
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zaz_call_function.

* Declaração de Variáveis
DATA: t_dados    TYPE TABLE OF string,
      v_dest     TYPE char10,
      v_taskname TYPE char10.

* Nome do DESTINATION (no caso aqui, apontado para o próprio sistema)
v_dest     = 'NONE'.

* Rodando uma função em paralelo (3 tasks)
DO 3 TIMES.

  v_taskname = sy-index.

* Chamada da função abrindo várias tasks
  CALL FUNCTION 'Z_ZUMBI_PARALELO'
    DESTINATION v_dest
    STARTING NEW TASK v_taskname
    PERFORMING f_recebe_dados ON END OF TASK
    TABLES
      dados_selecionados    = t_dados
    EXCEPTIONS
      communication_failure = 1
      system_failure        = 2.

  IF sy-subrc <> 0.
    EXIT.
  ENDIF.

ENDDO.

*&---------------------------------------------------------------------*
*&      Form  F_RECEBE_DADOS
*&---------------------------------------------------------------------*
FORM f_recebe_dados USING taskname.

* Esse perform será executando quando as tasks da função
* Z_ZUMBI_PARALELO acabarem de rodar. Para pegar os resultados,
* ou seja, os valores que a função retornou, use o comando abaixo:
  RECEIVE RESULTS FROM FUNCTION 'Z_ZUMBI_PARALELO'
    TABLES
      dados_selecionados  = t_dados.

ENDFORM.                    " F_RECEBE_DADOS
```

Lembrando que no caso acima, eu coloquei o DESTINATION apontando para o mesmo sistema onde o programa está rodando. Nada impede que você abra tasks num sistema destino. Ah, e o código acima, obviamente, não faz muito sentido funcionamente... é mais para mostrar os comandos mesmo. 🙂

IN BACKGROUND TASK: Muito parecido com o UPDATE TASK, no COMMIT WORK um programa que roda num Work Process em Background dispara a execução da(s) função(s). Ele é pouco usado em programas Z, mas o standard usa bastante para executar processos gigantes pós-commit das transações.

Phew, terminei os mais usados. Aidna tem muitos outros tipos de chamadas de função, mas acho que esses são os mais usados para programas Z.

Abraços a todos aqueles que já chamaram uma função.

Comentários

> **Eduardo Ribeiro** — *23/08/2025 13:44*
>
> Importante lembrar que a função precisa existir no sistema destino...

> **Henrique Dias** — *22/08/2013 15:12*
>
> Uma coisa interessante que eu descobri esses dias é que você pode colocar a exceção ERROR_MESSAGE em uma função para suprimir as mensagens que ela exibiria. Muito útil para algumas funções metidas a bapi que mostram mensagem invés de usar tabela de retorno.

> > **igor vilela** — *06/09/2013 15:39*
> >
> > so pra constar ... interessante de mais seu comentario ... porem explicar melhor ...
> > Ex.
> > CALL FUNCTION 'GUI_DOWNLOAD'
> > EXPORTING
> > FILENAME = LV_DIR
> > TABLES
> > DATA_TAB = LT_DATA
> > EXCEPTIONS
> > ERROR_MESSAGE = 0
> > OTHERS = 22.

> > > **Henrique Dias** — *24/09/2013 09:57*
> > >
> > > Usando seu exemplo ficaria assim:
> > >
> > > CALL FUNCTION 'GUI_DOWNLOAD'
> > > EXPORTING
> > > FILENAME = LV_DIR
> > > TABLES
> > > DATA_TAB = LT_DATA
> > > EXCEPTIONS

> ERROR_MESSAGE = 1
> OTHERS = 2.

> **Henrique Dias** — *24/09/2013 10:18*
>
> A última mensagem apresentada pela função vai atualizar as variáveis SY-MSGID, SY-MSGTY, SY-MSGNO, SY-MSGV1, SY-MSGV2, SY-MSGV3 e SY-MSGV4.

> **Thiago** — *10/12/2018 08:08*
>
> Boa Henrique! Excelente dica!
> Já estava quase fazendo todo um paranauê pra rodar uma função em background por causa de uma mensagem de erro. Valeu!

Kaio — *08/04/2011 08:59*

caso eu queira criar uma função 'z' e colocar algumas exceções quais codigos eu preciso inserir no texto fonte da função?!

grato

> **Mauricio Cruz** — *11/04/2011 07:11*
>
> Fala Kaio!
>
> Primeiro você deve "declarar" a existência da exceção na aba "Exceptions" da SE37. Depois, você deve seguir o mesmo padrão de declaração ali do exemplo da chamada da função GUI_UPLOAD. E lembre-se de disparar a exceção dentro da função com o comando RAISE EXCEPTION, sempre que tiver um erro na execução da função.
>
> Abraços 🙂

Rafael Vieira — *24/03/2011 06:42*

mano, vc é um lixo

"Zumbis estão atacando" foi foda
kkkkkkkkkkkkkkkkkkkk

O PERFORMING ON END OF TASK é útil hein, não conhecia isso não.
Abraço!

> **Mauricio Cruz** — *24/03/2011 06:51*
>
> houeahehaoehaoh
>
> tem que zuar, senão fica chato demais, pô 😀

> abraço!