



Debugger

SAP



SAP/ABAP Debugger

(Antigo e Novo)

Debugger Antigo

- 1 Introdução
- 2 Introdução ao código ABAP
- 3 Ferramentas básicas
- 5 Dicas e truques

Debugger Novo

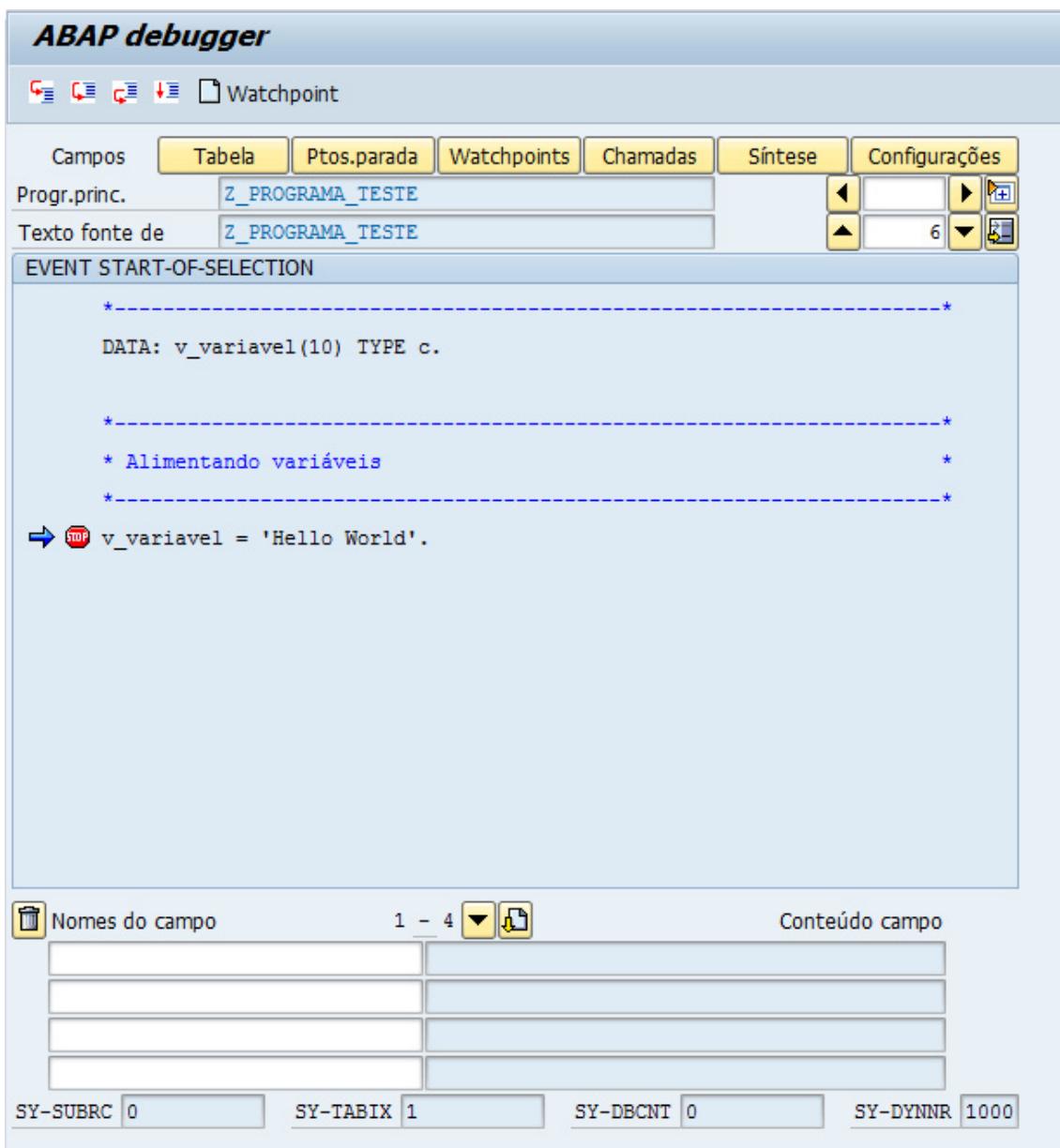
- 1 Introdução
- 3 Ferramentas básicas
- 4 Dicas e truques

Introdução ao Debugger Antigo

O online debugger é uma ferramenta para diagnosticar problemas com o código do programa.

Ele permite a execução passo a passo de cada etapa que o programa executa, permitindo entender como um programa funciona, a encontrar problemas ou até mesmo alterar os dados de execução.

É necessário ter uma leve noção de código ABAP para que se possa entender como o Debug funciona, vamos visualizar este ponto no capítulo 2, a fim de entender o básico para melhor gerenciamento da ferramenta.



The screenshot shows the SAP ABAP debugger interface. At the top, there's a toolbar with icons for back, forward, and search, followed by a "Watchpoint" button. Below the toolbar, a navigation bar has tabs for "Campos", "Tabela" (which is selected), "Ptos.parada", "Watchpoints", "Chamadas", "Síntese", and "Configurações". Underneath the navigation bar, there are two input fields: "Progr.princ." containing "Z_PROGRAMA_TESTE" and "Texto fonte de" also containing "Z_PROGRAMA_TESTE". To the right of these fields are several small buttons for navigating through the code. The main area displays ABAP code:

```
EVENT START-OF-SELECTION
  DATA: v_variavel(10) TYPE c.

  *-----*
  * Alimentando variáveis
  *-----*
  → STOP v_variavel = 'Hello World'.
```

At the bottom of the code editor, there's a section titled "Nomes do campo" (Field Names) with a dropdown menu showing "1 - 4" and a "Delete" icon. To the right of this is a "Conteúdo campo" (Field Content) section with four empty rows. At the very bottom, there are four input fields with values: SY-SUBRC [0], SY-TABIX [1], SY-DBCNT [0], and SY-DYNNR [1000].

Introdução ao código ABAP (1ª parte)

Para o melhor entendimento de um Debug, é necessário que se conheça alguns comandos básicos em ABAP, a fim de identificar cada passo e ponto que o programa está executando, também é interessante saber como funcionam os operadores de condição, para não se confundir na hora de realizar a análise, abaixo estão alguns comandos simples e suas respectivas funções:

Todo programa é separado em eventos, temos desde a tela de seleção ao fim do relatório, bapi, batch-input, etc.

- **Tela de seleção**

Na tela de seleção podemos ter dois tipos de códigos para a declaração de botões, o Parameter e o Select-options.

Parameter: é usado para declarar um botão com seleção única na tela de seleção, podem haver inúmeros parameters, todos são codificados conforme abaixo:

```
PARAMETERS: p_parameter(10) TYPE c.
```

O comando PARAMETERS é fixo, faz referência à chamada da função que cria o botão, já o nome p_parameter é o nome fornecido pelo usuário, acompanhado de seu tamanho e referência, o comando TYPE é o que indica ao que ele irá fazer referência, no caso, pode se fazer referência direta a um campo da tabela, por exemplo, TYPE mara-matnr.

Select-options: Tem a mesma função do parameter, porém permite fazer a seleção múltipla e com condições na tela de seleção, todos são codificados conforme abaixo:

```
SELECT-OPTIONS: s_select FOR mara-matnr.
```

O comando SELECT-OPTIONS é fixo, faz referência a chamada da função que cria o botão, já o nome s_select é fornecido pelo usuário, acompanhado de sua referência “FOR” que indica a qual campo ele será estruturado para a seleção múltipla.

- **Variáveis**

As variáveis são caixas de memória que guardam valores dentro de um programa, elas podem carregar qualquer tipo de valor, dependendo da sua declaração, vejamos um exemplo abaixo:

```
DATA: v_variavel(11) TYPE c.
```

O comando DATA serve para criar um objeto, ele é utilizado para várias declarações de objetos, a diferença sempre está no meio “TYPE”, que é o que define qual será o objeto a ser criado, o nome v_variavel é definido pelo usuário, assim como seu tamanho e tipo.

- **Tabelas Internas**

As tabelas internas são tabelas de memória que guardam valores de uma seleção ou de uma execução no programa, elas contêm uma estrutura parecida com as transações SE16, porém são exibidos apenas os nomes técnicos dentro das mesmas, vejamos um exemplo da codificação de uma tabela interna abaixo:

```
DATA: t_mara TYPE TABLE OF mara.
```

Como dito anteriormente, o comando DATA indica a criação de um objeto no programa, o nome “t_mara” é definido pelo usuário para dar nome a tabela, o comando TYPE TABLE OF é o que define que esse objeto é uma tabela referenciada a tabela que se encontra a frente “mara”.

É possível criar uma tabela baseada em uma estrutura dentro do programa, usando o comando TYPES, vejamos um exemplo de como é realizada abaixo:

```
TYPES: BEGIN OF ty_mara,
        matnr TYPE mara-matnr,
        wrkst TYPE mara-wrkst,
    END OF ty_mara.
```

```
DATA: t_mara TYPE TABLE OF ty_mara.
```

- **Estruturas**

Uma estrutura é uma tabela de apenas uma linha, pode ser utilizada para a leitura das linhas de uma tabela interna, ou apenas para o preenchimento de vários campos, pode ser comparada também a variáveis, porém todas dentro de um mesmo objeto, se definirmos objetos no TYPES e em seguida criarmos uma estrutura, teremos um objeto com várias variáveis que podem guardar valores para a execução no programa, vejamos o exemplo de sua codificação abaixo:

```
DATA: e_mara TYPE mara.
```

O comando TYPE faz referência a tabela Mara, criando uma estrutura com todos os seus campos, ou seja, todos os campos são como variáveis dentro de um único objeto, fazendo referência a uma tabela.

Também é possível criar uma estrutura baseada em uma tabela criada no programa, seguindo o mesmo exemplo da tabela interna, veja seu código abaixo:

```
TYPES: BEGIN OF ty_mara,
        matnr TYPE mara-matnr,
        wrkst TYPE mara-wrkst,
    END OF ty_mara.
```

```
DATA: e_mara TYPE ty_mara.
```

- **Ranges**

Os ranges são intervalos de valores usados dentro de um programa, eles contém uma estrutura parecida com o SELECT-OPTIONS e permitem que você carregue intervalos com diferentes condições dentro de um código, vejamos abaixo um exemplo da codificação de um range:

```
DATA: r_range TYPE RANGE OF mara-matnr.
```

O comando TYPE RANGE OF indica que o objeto será um range baseado em uma referência “mara-matnr”.

O range contém valores operacionais, tais como:

EQ, NE, BT, etc., esses valores serão explicados em outro tópico.

- **Constantes**

Uma constante é um valor fixo dentro do programa, porém facilita a edição de valores fixos, diferenciando-a de um hard code, que seria a inserção de um código diretamente em uma variável, tabela, etc.

Por exemplo:

A constante é declarada da seguinte forma:

```
CONSTANTS: c_constante(11) TYPE c VALUE 'Hello Word'.
```

Tendo a constante declarada, podemos passá-la quantas vezes precisarmos a vários pontos do programa:

```
v_variavel = c_constante.
```

Quando usamos um hard code, temos que digitar para cada utilização todo o seu valor, também implicando na troca de todos os valores, no caso da constante, bastaria mudar apenas o valor depois do comando “VALUE” e todos os objetos que fazem referência a constante seriam afetados.

```
v_variavel = 'Hello World'.
```

- **Blocos**

Sempre que passamos a tela de seleção, definimos o programa em blocos, o que facilita a organização dos códigos dentro de um código fonte, esses blocos são chamados de PERFORMS.

Um PERFORM quando codificado gera um FORM, que é a sua sub sequência, a chamada PERFORM pode se fazer referência a diferentes FORMS, já que o código PERFORM é apenas o código que dá função a sequência de blocos, vejamos um exemplo abaixo:

```
PERFORM: f_form1,  
         f_form2.
```

```
FORM f_form1.
```

```
ENDFORM.
```

```
FORM f_form2.
```

```
ENDFORM.
```

O comando PERFORM está chamando dois FORMs, que estão criados logo abaixo dos mesmos, sempre que o ponteiro do debug passar por uma dessas chamadas, ele cairá dentro de um FORM, e assim que terminar, seguirá a sequência para o programa ou para o próximo FORM.

- **Operadores**

Um operador define qual será a condição aplicada a uma lógica, eles podem ser usados de forma simples, usando os operadores tradicionais ou por comandos ABAP, ambos com a mesma função, são eles:

- = ou EQ: - Igual a.
- <> ou NE - Diferente de.
- <, LT - Menor que.
- >, GT - Maior que.
- <=, LE - Menor ou igual a.
- >=, GE - Maior ou igual a.

- **Seleções**

Quase todos os programas do SAP são criados com base em seleções em tabelas transparentes, dentro de um código, podemos encontrar diversas seleções de diferentes formas, porém vejamos o exemplo e os detalhes de uma seleção simplificada:

```
SELECT *
FROM mara
INTO TABLE t_mara
WHERE matnr EQ v_matnr.
```

O comando SELECT dá forma ao código iniciando uma seleção, você pode informar * para todos os campos ou informar quais os campos que deseja selecionar, o comando FROM indica de qual tabela serão selecionados os campos, o comando INTO TABLE define em qual tabela interna serão aplicados os campos assim como o comando WHERE diz quais serão as condições para a seleção, podem haver mais de uma condição, elas também podem ser diferentes, baseadas nos operadores.

- **Leituras**

As leituras são realizadas para vermos todos os valores ou um único valor de uma tabela, temos dois tipos principais de leituras, o LOOP e o READ TABLE, vejamos os detalhes de cada um:

LOOP - É utilizado para ler os valores de uma tabela interna, passando por todos os valores baseados ou não em uma condição (WHERE), vejamos abaixo um exemplo do código de um LOOP:

```
LOOP AT t_mara INTO e_mara.

ENDLOOP.
```

Entre o LOOP e ENDLOOP será aplicada a codificação que se faz necessária para toda as linhas da leitura, o comando INTO indica em que estrutura será lida cada linha do LOOP, no caso “e_mara”, o LOOP realizado desta maneira irá ler todas as linhas preenchidas.

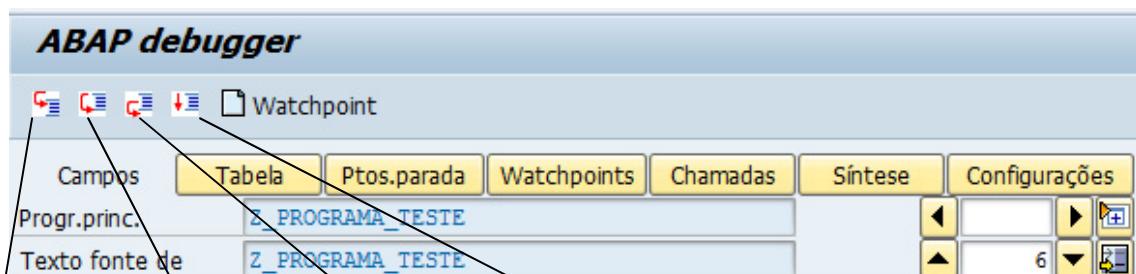
READ TABLE - É utilizado para ler uma única linha de uma tabela, dentro ou fora de um LOOP, é extremamente essencial quando temos uma tabela transparente que tem cabeçalho e item, onde lemos o cabeçalho no LOOP e cada item dentro do READ TABLE, vejamos abaixo um exemplo do código de um READ TABLE:

```
READ TABLE t_mara INTO e_mara WITH KEY matnr = v_matnr.
```

Se o valor da condição WITH KEY for encontrado, a estrutura “e_mara” terá a linha selecionada pela condição, se o valor não for atendido, a estrutura ficará vazia.

Ferramentas Básicas

Vejamos abaixo algumas ferramentas da tela inicial do Debug

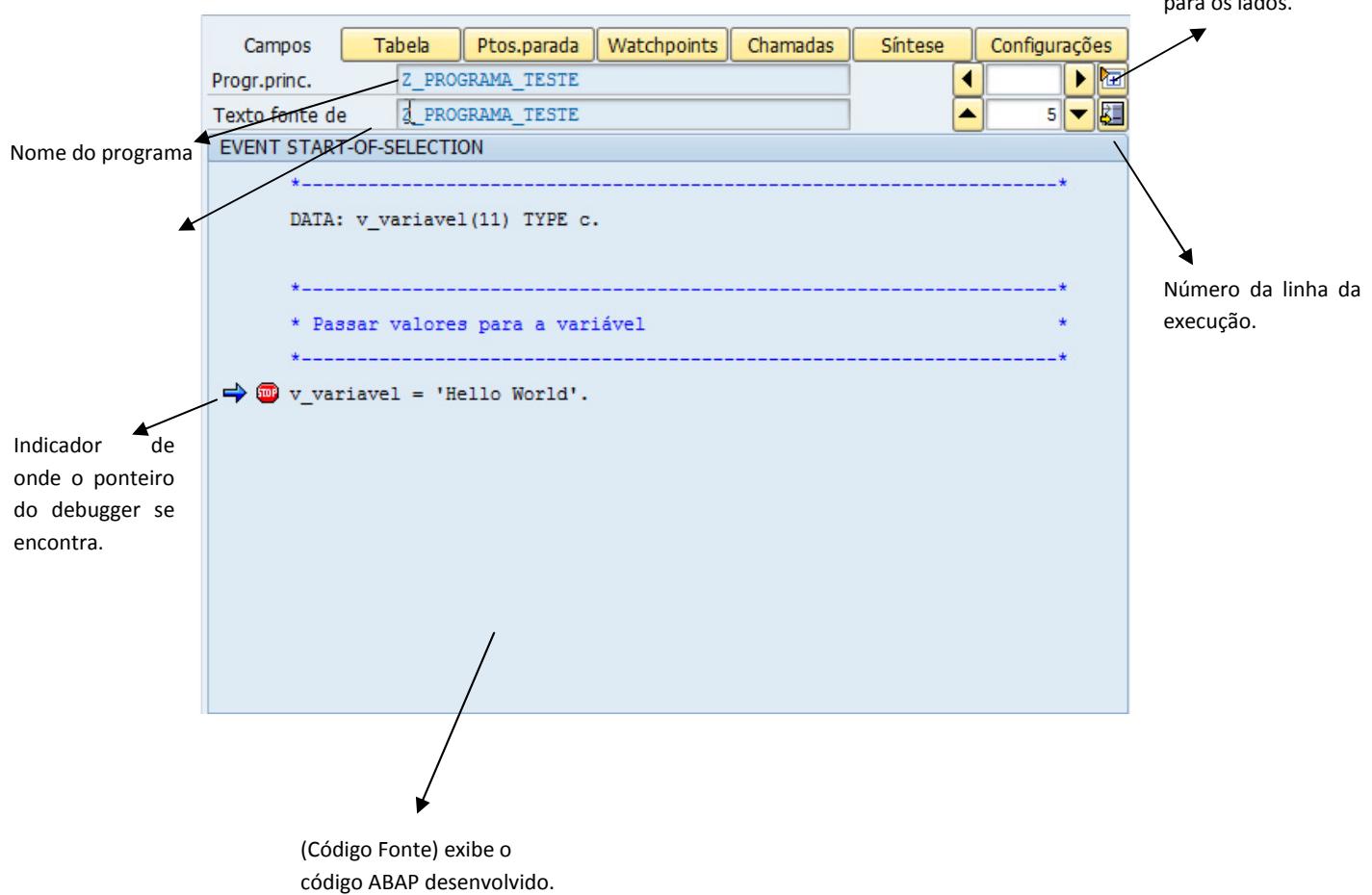


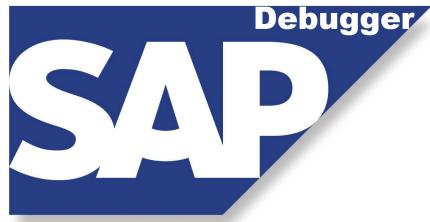
Avança uma etapa), ou seja, lê a próxima linha do código (F5).

(Executar) faz com que o processo vá para a próxima linha mesmo que forçando-a (F6).

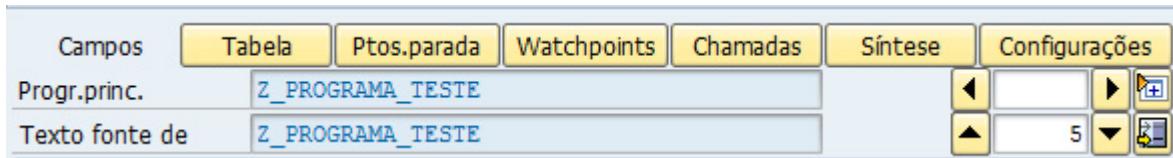
▲ (Avançar) executa o processo até que seja encontrada uma nova para ou o fim da execução (F8).

(Retorno) avança até a saída de um bloco, retornando para o programa principal (F7).





O botão abaixo contém todas as outras funções do Debug que vamos ver passo a passo:



- **Botão “Campos”:**

Contém toda a estrutura dos campos que selecionamos para exibição e modificação durante o Debug, todos os campos estarão sendo exibidos abaixo do código fonte, conforme a tela abaixo:



- **Botão “Tabela”:**

Contém a estrutura para a visualização e modificação dos campos das tabelas internas, nesta opção também é possível inserir e eliminar linhas, os detalhes de cada botão estão no exemplo abaixo:

Veja que os campos exibidos abaixo são editáveis, permitindo assim apagar o valor de um campo da tabela e inserir outro, conforme no exemplo abaixo:



Apague o valor ERSDA e pressione ENTER note que o valor do campo sumiu, porém ele ainda está na tabela, esse controle serve apenas para a exibição dos campos que você deseja, ou a ordem que preferir.

Tabela interna		t_mara	ERSDA	ERNAM	LAEDA	AENAM	VPSTA	
1	MATNR		ERSDA	ERNAM	LAEDA	AENAM	VPSTA	
1		0000000<		00000000				...

Tabela interna		t_mara	ERSDA	ERNAM	LAEDA	AENAM	VPSTA	
1	MATNR		ERSDA	ERNAM	LAEDA	AENAM	VPSTA	
1				00000000				...

- **Botão “Ptos.Parada”:**

Exibe todos os pontos de paradas que estão ativos no programa, os detalhes estão no exemplo abaixo:

Ptos.parada			
Nº	Tipo ponto parada		Em (caminho absol.)
STOP	1 Ponto de programa	Z_PROGRAMA_TESTE (18)	
STOP	2 Ponto de programa	Z_PROGRAMA_TESTE (20)	
STOP	3 Ponto de programa	Z_PROGRAMA_TESTE (22)	
STOP	4 Ponto de programa	Z_PROGRAMA_TESTE (24)	
STOP	5 Ponto de programa	Z_PROGRAMA_TESTE (28)	
STOP	6 Ponto de programa	Z_PROGRAMA_TESTE (26)	
STOP	7 Ponto de programa	Z_PROGRAMA_TESTE (30)	
STOP	8 Ponto de programa	Z_PROGRAMA_TESTE (32)	



- **Botão “Chamadas”:**

O botão chamado tem como objetivo indicar o bloco ativo na execução, por exemplo, se o programa entrar em um FORM, esta opção exibirá que o processo está dentro de um Form, ou dentro de dois, indicando em quantos processos o programa se aprofundou, ótimo para quando existem várias chamadas de Perform dentro de um programa, facilitando para que o usuário não se perca de onde o ponteiro está passando, abaixo seguem os detalhes dessa ferramenta:



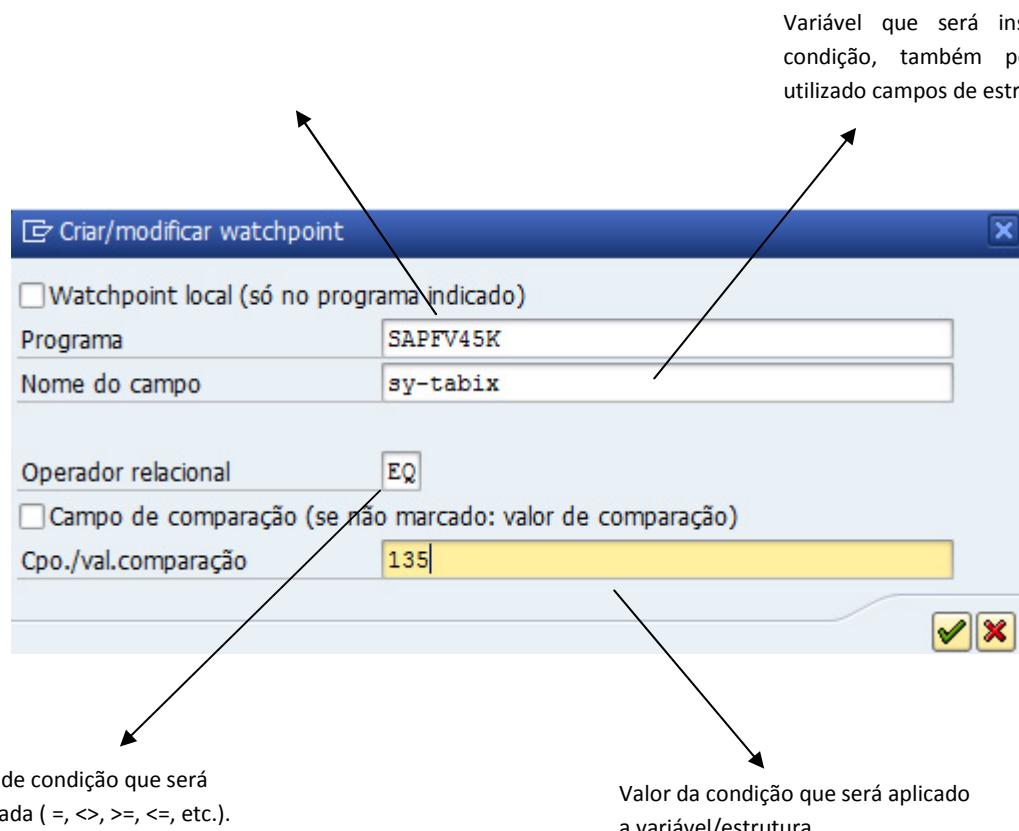
- **Botão “Síntese”:**

O botão síntese tem como objetivo demonstrar por quais programas o indicador do debugger passou, sendo assim, em uma transação standard, por exemplo, você saberá quais são os pontos e programas utilizados até então, facilitando a volta em passos anteriores para recuperar valores, abaixo os detalhes desta ferramenta:

- **Botão “Watchpoint”:**

O Watchpoint tem como objetivo inserir uma condição para um ponto de parada, por exemplo, se você quer que dentro de um Loop o programa pare exatamente na linha 135, esta será a opção que te ajudará, caso contrário você terá que passar linha a linha no Debug, perdendo grande tempo de análise, também é possível criar um ponto de parada baseado no preenchimento de uma variável, ou quando ela fica vazia, esta opção é muito importante para poupar tempo de análise e ser objetivo na condição que você deseja, abaixo seguem os detalhes dessa ferramenta:

Primeiro criamos o Watchpoint com o botão que fica acima da barra de botões >





O botão  serve para monitorar todos os Watchpoints que foram criados até o momento, podendo eliminar ou editar algum que não se faça mais necessário, veja os detalhes no exemplo abaixo:



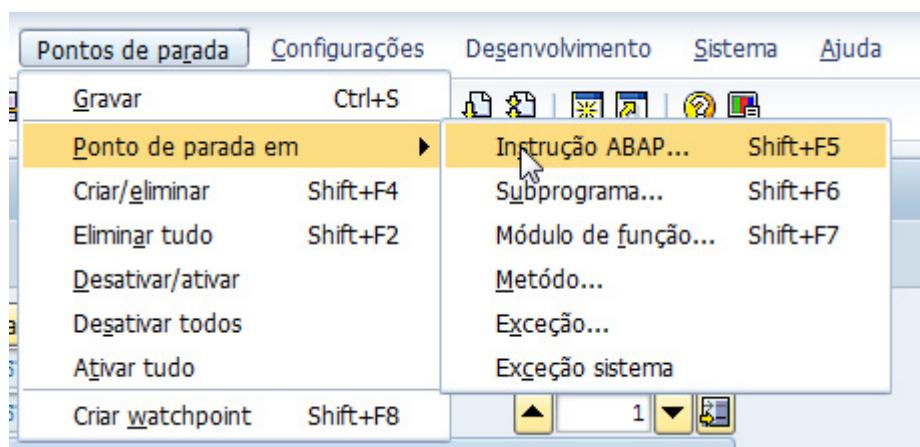
Dicas e truques

A principal utilidade de um Debug são suas ferramentas adicionais, que permitem que você encontre algumas funções no SAP como EXIT, ENHANCEMENT, BADIs, Avance, Volte, dentre outros.

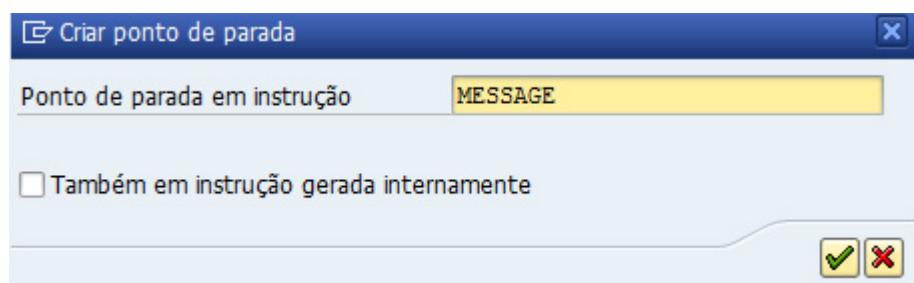
Vejamos abaixo algumas dessas dicas usadas no dia a dia.

- **Parando em pontos específicos**

O SAP permite que o Debug avance até pontos específicos, como métodos, funções, dentre outros, existe uma opção que contém todas essas funções, vejamos abaixo como executá-la:



Instrução ABAP: Permite criar Break-points em uma codificação específica, muito útil para encontrar, por exemplo, mensagens de sucesso ou erro, para isso basta digitar o nome do comando na caixa, conforme abaixo, que o ABAP irá criar pontos de parada para cada parte do programa que contiver esse código:



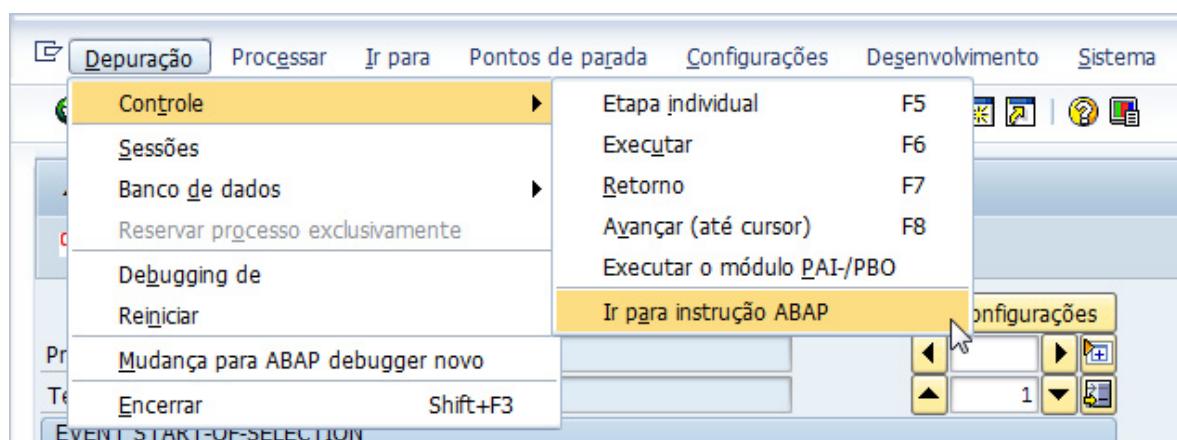
Essa opção serve para qualquer código ABAP, basta informar o código na caixa e clicar em OK.

As demais opções como Subprograma, Módulo de Função, Método, Exceção, Exceção do sistema se aplicam a mesma lógica explicada acima, porém só funcionam para seus objetos específicos, se por exemplo, você precisar parar em uma função, vá até a opção Módulo de Função e insira o nome da função, o ABAP criará um ponto de parada unicamente na função informada.

Debugger

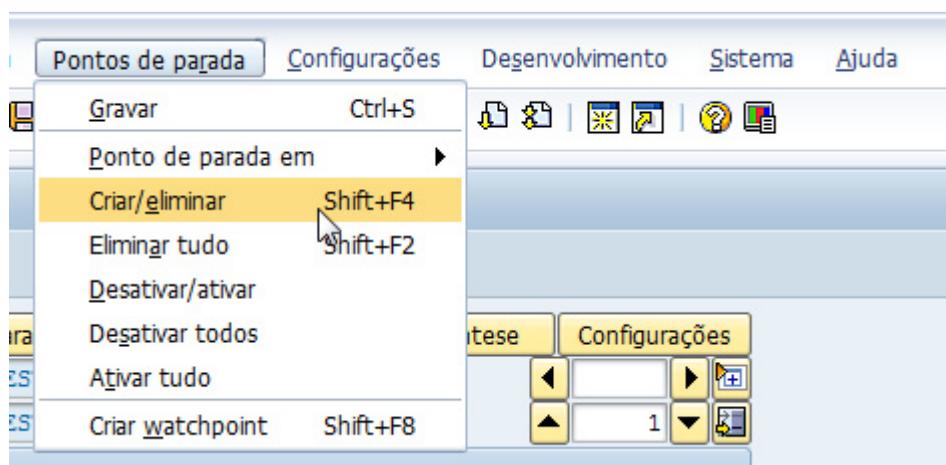
- **Ir para instrução ABAP**

É possível voltar ou avançar para um ponto no programa, basta clicar em cima do ponto e seguir conforme a imagem abaixo, lembrando que alguns desses “saltos” não são permitidos, pois alguns pontos no programa serão obrigatórios de serem passados, mas é útil por exemplo para voltar para antes de uma mensagem, mudar o seu valor e voltar novamente se necessário, ou até mesmo pular um erro e continuar o processo.



- **Desativar/Ativar/Eliminar pontos de parada**

Quando pontos de parada são criados, é possível manipulá-los através da opção da imagem abaixo:



Ao clicarmos em eliminar, aquele ponto de parada selecionado será eliminado, se clicarmos em eliminar tudo, todos os pontos de parada criados serão excluídos, a opção Ativar/Desativar, permite que em uma execução os pontos não funcionem, porém continuem salvos dentro do programa, quando clicarmos em ativar novamente, eles voltarão a funcionar, o mesmo vale para a função Ativar todos e Desativar todos.

Debugger

- **Encontrando EXITs dentro de uma transação**

Através do Debug podemos encontrar EXITs utilizando apenas um comando simples, já anteriormente explicado, vejamos como fazer abaixo:

The screenshot shows the SAP ABAP debugger interface. The title bar says "ABAP debugger". Below it is a toolbar with icons for back, forward, and search. A menu bar has tabs: Campos, Tabela, Ptos.parada, Watchpoints, Chamadas, Síntese, and Configurações. The "Ptos.parada" tab is selected. In the main area, there's a table with "Progr.princ." set to "SAPMV45A" and "Nº tela" set to "0101". Below the table is a section labeled "PAI" with a blue arrow pointing right and the text "PROCESS AFTER INPUT.". The code shown is:
* Funktion mit richtigem Typ sofort ausführen
MODULE FUNKTION_AUSFUEHREN AT EXIT-COMMAND.

* Verkaufsorganisation prüfen
CHAIN.
FIELD : VBAK-VKORG,
TVKOT-VTEXT,
VBAK-UTWEG.
A modal dialog box titled "Criar ponto de parada" is open. It contains a text input field with "call customer-function" and a checked checkbox below it that says "Também em instrução gerada internamente". At the bottom are two buttons: a green checkmark and a red X.

O comando call customer-function cria pontos de parada em todas as chamadas das EXITs, assim que criado esse ponto de parada, o SAP irá parar em todas as Exits de todos os processos, então a desativação ou eliminação se faz necessária assim que a EXIT desejada for encontrada.

- **Encontrando Enhancements pelo Debugger**

Existem algumas maneiras de se encontrar um enhancement, como estamos nos tratando do Debug, a maneira mais adequada seria utilizar a mesma lógica aplicada para EXITs, porém essa não se faz tão eficaz, já que a SAP possui vários Enhancements Standards, abaixo segue o exemplo:

The screenshot shows the SAP ABAP debugger interface. At the top, there's a toolbar with icons for back, forward, and search. Below it is a menu bar with tabs: Campos, Tabela, Ptos.parada, Watchpoints, Chamadas, Síntese, and Configurações. The 'Ptos.parada' tab is selected, showing 'Progr.princ.' set to 'SAPMV45A' and 'Nº tela' set to '0101'. A PAI (Programmer's Analysis Interface) window is open, displaying the following code:

```
→ PROCESS AFTER INPUT.

* Funktion mit richtigem Typ sofort ausführen
MODULE FUNKTION_AUSFUEHREN AT EXIT-COMMAND.
```

Below the code, a sub-dialog titled 'Criar ponto de parada' (Create Breakpoint) is displayed. It contains a text input field 'Ponto de parada em instrução' with the value 'enhancement' highlighted in yellow. There's also a checkbox 'Também em instrução gerada internamente' (Also in internally generated instruction) which is unchecked. At the bottom right of the dialog are two buttons with checkmarks and crosses. At the very bottom of the main window, there's a row with a trash icon, 'Nomes do campo', a dropdown showing '1 - 4', and a 'Conteúdos campo' button.

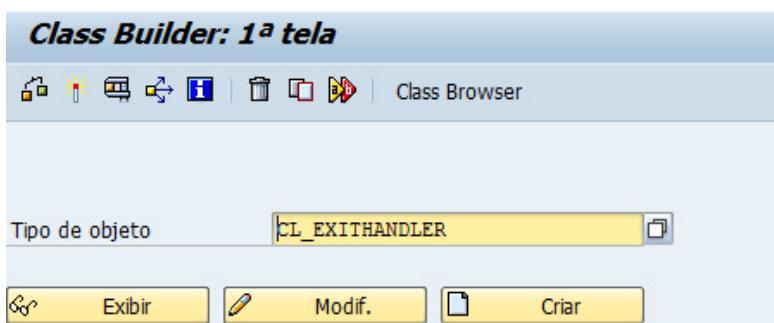
Debugger



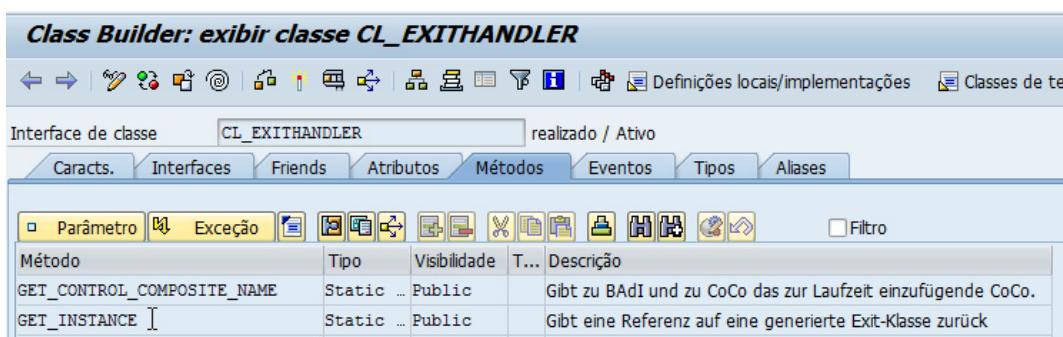
- Encontrando BADIs pelo Debugger

Diferente das outras duas formas, para encontrar uma BADI pelo Debugger devemos seguir os passos conforme abaixo:

Entrar na transação SE24 como objeto CL_EXITHANDLER:



Dar um duplo clique no método GET_INSTANCE



Marcar um ponto de parada conforme abaixo:

```
DATA: exit TYPE REF TO object,
      type_ref TYPE REF TO cl_abap_typedescr,
      class_name TYPE seoclsname,
      abs_type_classname TYPE string,
      def_Impl TYPE seex_boolean VALUE seex_false,
      mig_enhspotname TYPE enhspotname,
      is_Impl TYPE enhboolean,
      mig_badi_name TYPE enhbadiid.

| CALL METHOD cl_exithandler->get_class_name_by_interface
  EXPORTING
    instance                  = instance
  IMPORTING
    class_name                = class_name
  CHANGING
    exit_name                 = exit_name
  EXCEPTIONS
    no_reference              = 1
    no_interface_reference   = 2
    no_exit_interface        = 3
    data_incons_in_exit_managem = 4
    class_not_implement_interface = 5
    OTHERS                     = 6.
```



Debugger



Vejamos como exemplo a transação VA01, ao acessarmos o DEBUG será aberto, o parâmetro exit_name conterá o nome da BADI que está sendo passada naquele momento, veja na imagem abaixo:

ABAP debugger

Watchpoint

Campos Tabela Ptos.parada Watchpoints Chamadas Síntese Configurações

Progr.princ. CL_EXITHANDLER=====CP

Texto fonte de CL_EXITHANDLER=====CM001

METHOD GET_INSTANCE (CL_EXITHANDLER)

```
→ STOP CALL METHOD cl_exithandler=>get_class_name_by_interface
      EXPORTING
          instance = instance
      IMPORTING
          class_name = class_name
      CHANGING
          exit_name = exit_name
      EXCEPTIONS
          no_reference = 1
          no_interface_reference = 2
          no_exit_interface = 3
          data_incons_in_exit_managem = 4
          class_not_implement_interface = 5
          OTHERS = 6.
```

Nomes do campo 1 - 4 Conteúdo campo

exit_name	BADI_SD_SALES		

SY-SUBRC 0 SY-TABIX 2 SY-DBCNT 0 SY-DYNNR 0101

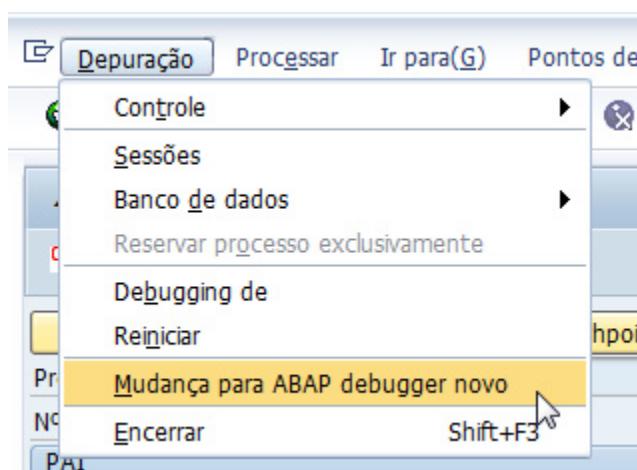
Introdução ao Debugger Novo

O online debugger novo é uma ferramenta para diagnosticar problemas com o código do programa.

Com algumas opções a mais, ele facilita a depuração do programa, permitindo um ambiente mais visual e mais agradável, também temos funções específicas para este Debug.

Para abrir o Debug novo, basta seguir a imagem abaixo:

Depuração/Mudança para ABAP debugger novo

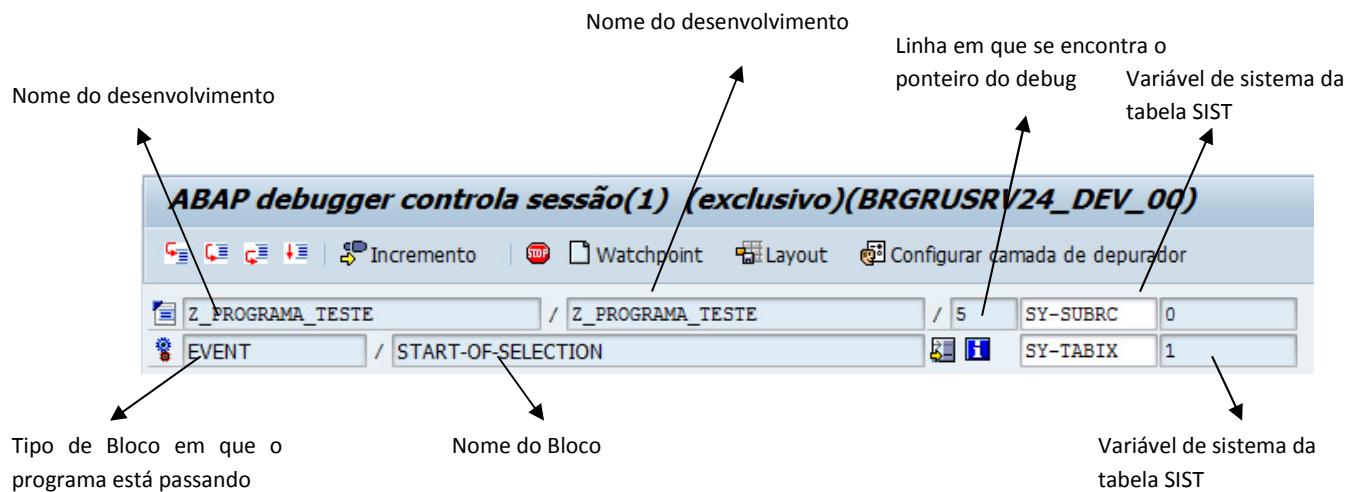


Irá carregar a tela abaixo:

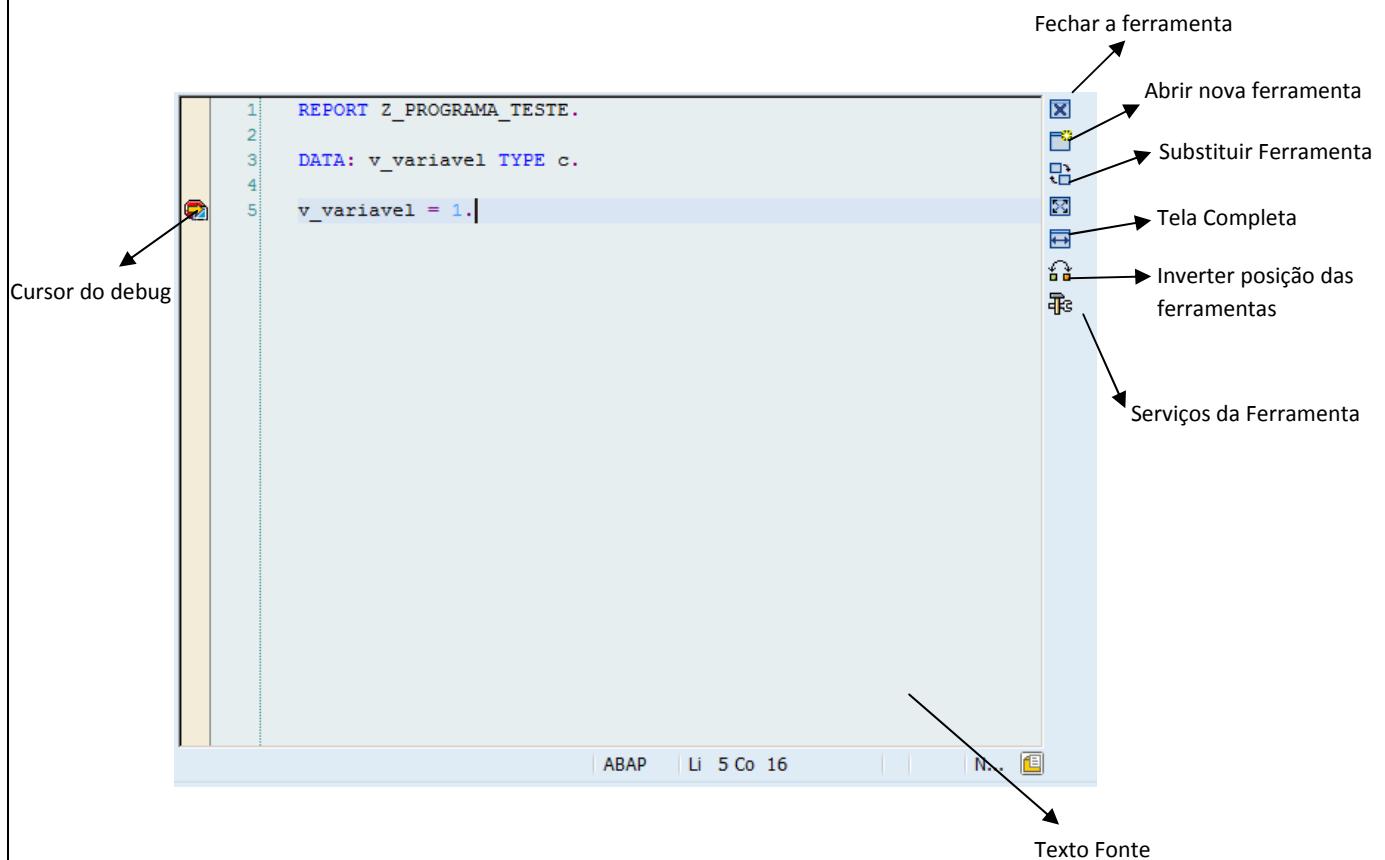
S...	Variável	C...	Val.	M...	Valor hexadecimal

Ferramentas Básicas

Vejamos abaixo, algumas ferramentas do novo Debugger detalhadamente:

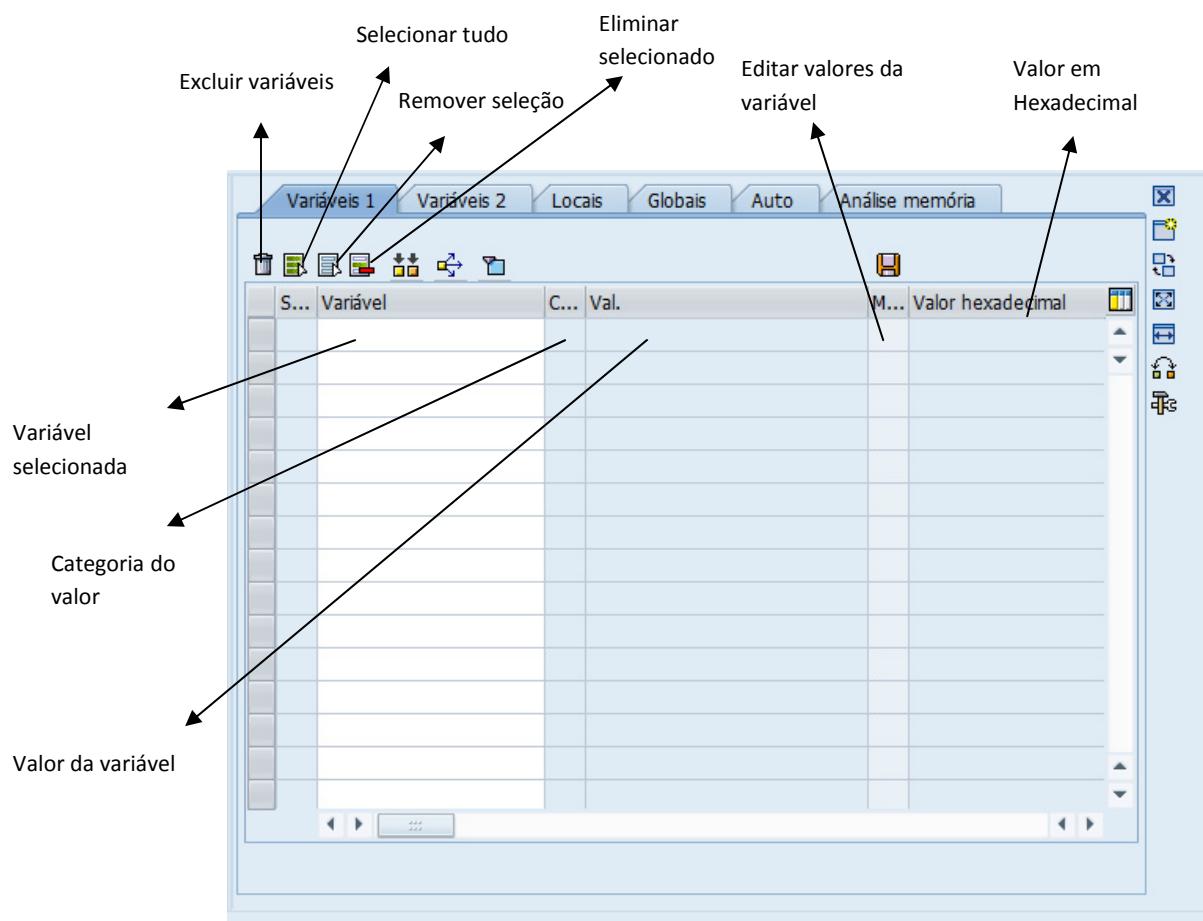


A visualização do texto fonte fica mais parecida com o desenvolvimento ABAP, destacando as palavras chaves para melhor entendimento, é possível também rolar a tela para cima e para baixo com o mouse.

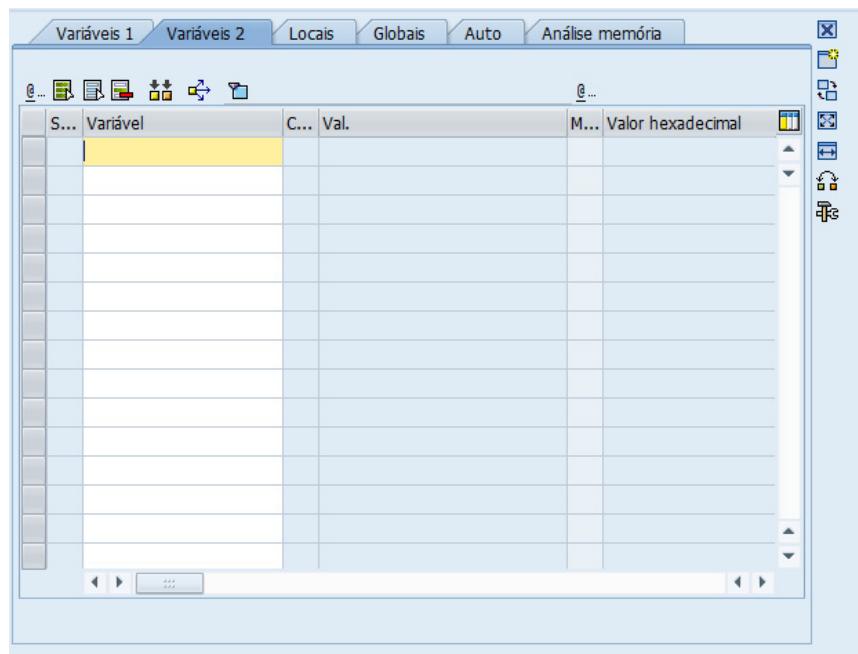




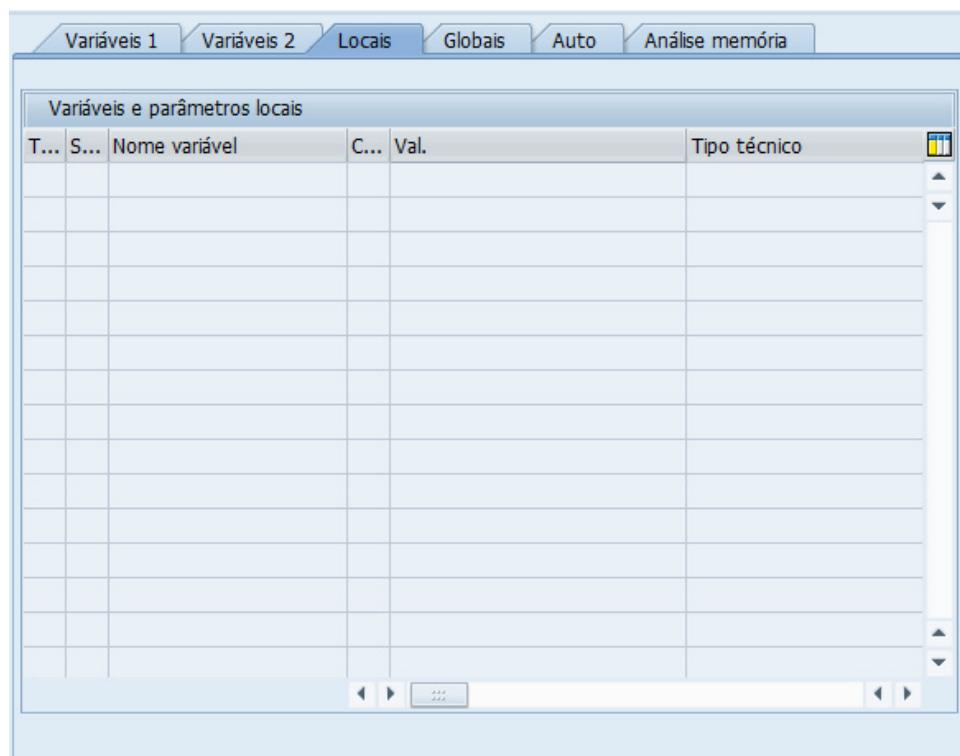
Esta é a tela inicial do Debug, similar a outra primeira tela do Debugger antigo, como ela é uma das telas principais, vamos explorar suas opções para depois darmos continuidade as outras ferramentas do Debug Novo:



A Aba de variáveis 2 permite que você selecione mais variáveis em um segundo plano, contém as mesmas funções da tela anterior:



A Aba Locais, mostra todos objetos declarados dentro de um bloco, muito útil ao passar por um PERFORM desconhecido e ver quais são os objetos e os valores de seus objetos, pode ser usado principalmente com debug standard, para encontrar valores para Badis, enhancements, exits, etc.





Debugger

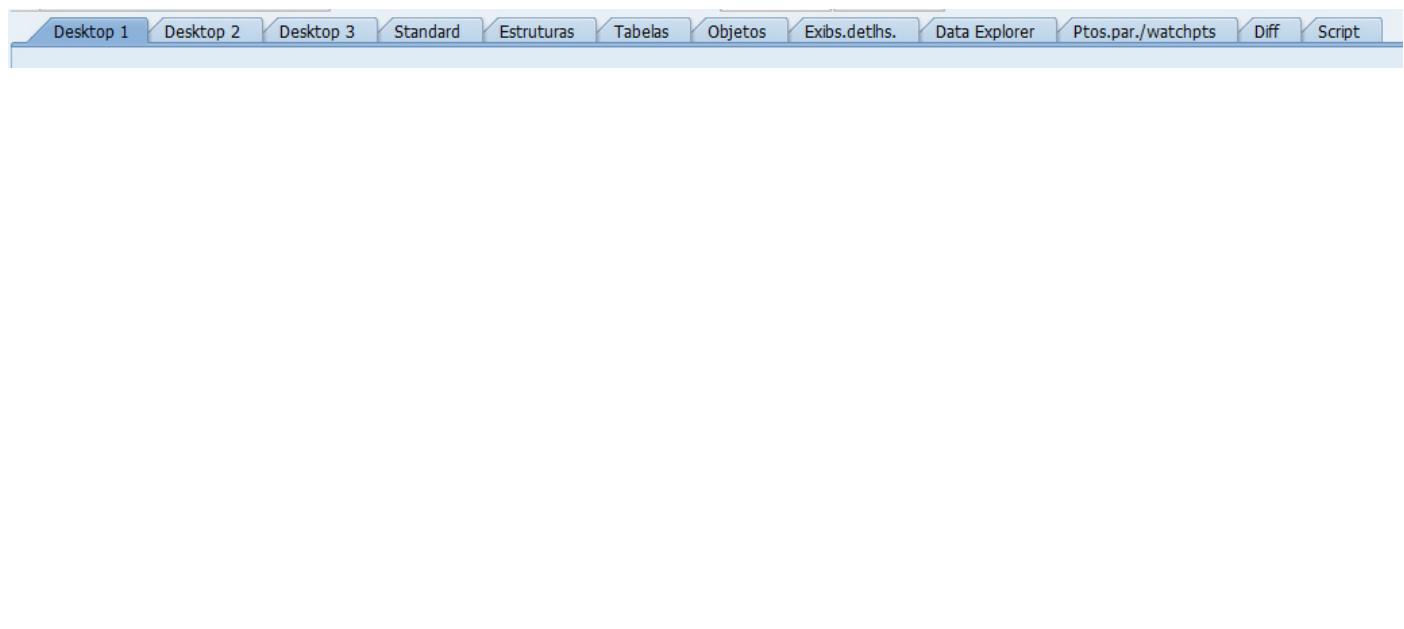


A Aba Globais tem a mesma função da locais, porém não somente para os objetos locais, mas sim para todos os objetos declarados no programa, também muito útil para encontrar exits, badis, etcs.

Variáveis globais					
T...	S...	Nome variável	C...	Val.	Tipos técnicos
		CVBAK		Structure: flat, not cha...	Structure: flat, not charli...
		CVBRK		Structure: flat, not cha...	Structure: flat, not charli...
		VVBRK		Structure: flat, not cha...	Structure: flat, not charli...
		CVBAP		Structure: flat, not cha...	Structure: flat, not charli...
		CFPLA		Structure: flat, not cha...	Structure: flat, not charli...
		CFPLT		Structure: flat, not cha...	Structure: flat, not charli...
		CVBRP		Structure: flat, not cha...	Structure: flat, not charli...
		VVBRP		Structure: flat, not cha...	Structure: flat, not charli...
		CVBEP		Structure: flat, not cha...	Structure: flat, not charli...
		CVBFA		Structure: flat, not cha...	Structure: flat, not charli...
		CVBKD		Structure: flat, not cha...	Structure: flat, not charli...
		CVBPA		000000	Structure: flat, charlike(5..)
		CVBSN		Structure: flat, not cha...	Structure: flat, not charli...
		CVBUP		000000	Structure: flat, charlike(6..)
		CVBUK			Structure: flat, charlike(1..)

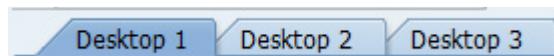
As Abas não mencionadas são desnecessárias para uma visualização básica do Debug, iremos seguir apenas no que se faz mais necessário para o entendimento básico do ABAP Debugger Novo.

A primeira tela que vimos contém algumas ferramentas, porém, temos uma sequência de abas logo acima que também contém novas opções com mais ferramentas, vamos explorar passo a passo cada uma delas:





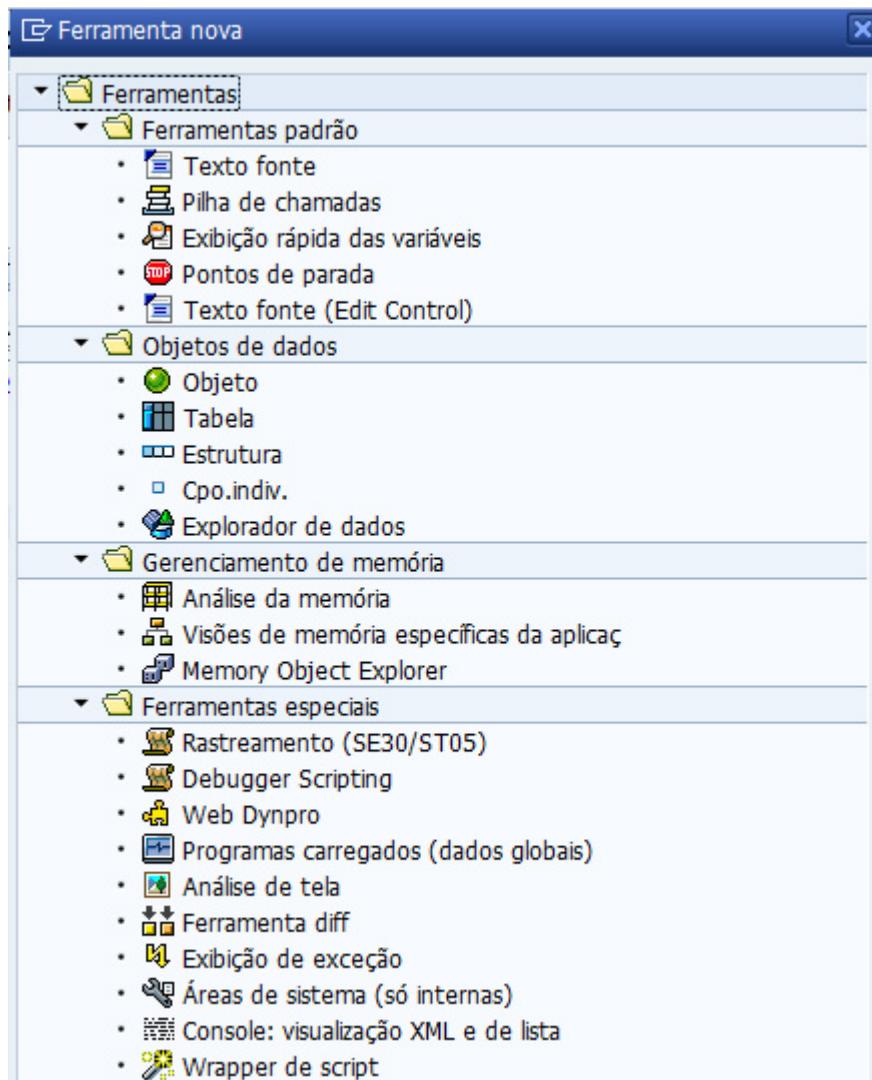
- Abas “Desktop 1, Desktop 2 e Desktop3”



As abas Desktop 1, Desktop 2 e Desktop 3 são editáveis, você pode configurar cada uma delas da forma que achar melhor, usando a opção ferramenta nova:



Com essas opções, você pode incluir ou remover novas ferramentas ao debugger, as que já estão visíveis normalmente ao iniciar o Debug Novo são: Texto Fonte e Exibição rápida das variáveis, como exploramos logo acima.



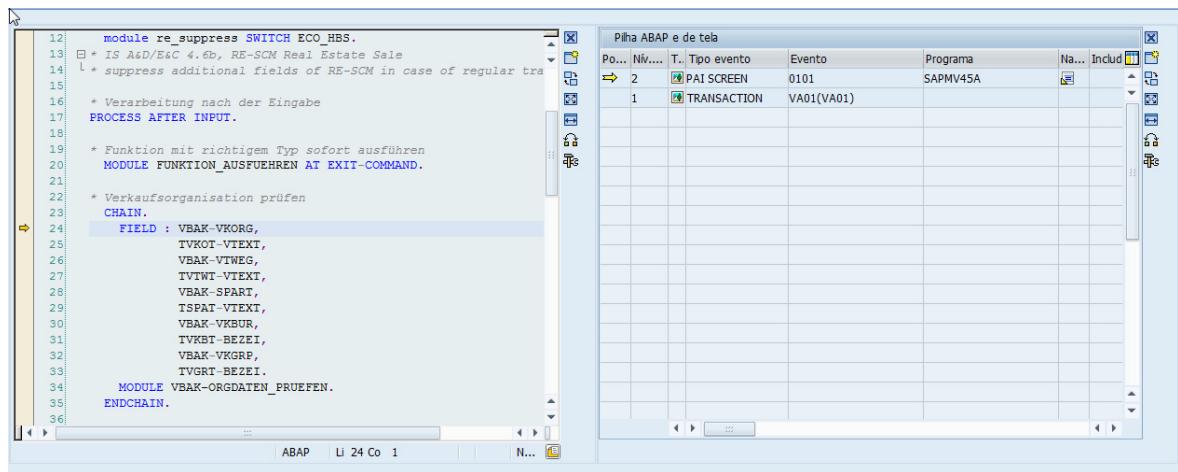
Debugger



• Aba Standard



A Aba standard vem com as ferramentas texto fonte e pilha ABAP e de tela preenchidas, também é possível editá-la, mas ela é muito útil na identificação dos passos por onde o programa passou, muito similar a opção “síntese” do debug antigo.



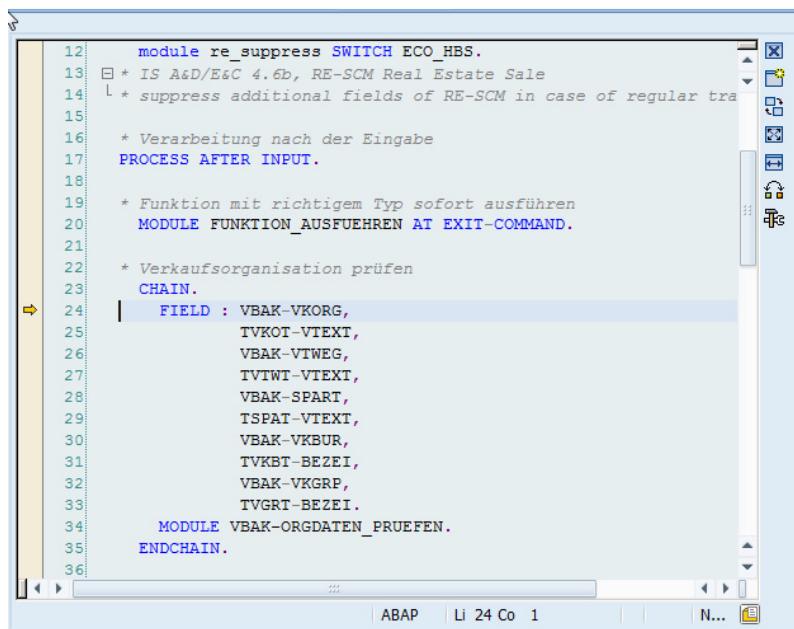
The screenshot shows the SAP ABAP Debugger interface in Standard mode. On the left is the ABAP code editor with the following content:

```
12 module re_suppress SWITCH ECO_HBS.
13   * IS A&D/E&C 4.6b, RE-SCM Real Estate Sale
14   * suppress additional fields of RE-SCM in case of regular tra
15
16   * Verarbeitung nach der Eingabe
17   PROCESS AFTER INPUT.
18
19   * Funktion mit richtigem Typ sofort ausführen
20   MODULE FUNKTION_AUSFUEHREN AT EXIT-COMMAND.
21
22   * Verkaufsorganisation prüfen
23   CHAIN.
24     FIELD : VBAK-VKORG,
25       TVKOT-VTEXT,
26       VBAK-VTWEG,
27       TWTW-TVTEXT,
28       VBAK-SPART,
29       TSPAT-VTEXT,
30       VBAK-VKBUR,
31       TVKBT-BEZEI,
32       VBAK-VKGGRP,
33       TVGRT-BEZEI.
34   MODULE VBAK-ORGDATEN_PRUEFEN.
35
36 ENDCHAIRN.
```

On the right, there is a table titled "Pilha ABAP e de tela" (Call Stack) with the following data:

Po...	Niv...	T...	Tipo evento	Evento	Programa	Na...	Inclu...
2			PAI SCREEN	0101	SAPMV45A		
1			TRANSACTION	V001(VA01)			

O editor do texto fonte mostra onde o ponteiro do debug se encontra, caso selecionado um outro momento o ponteiro ficará logo na chamada desse bloco, facilitando a visualização de outros pontos de um programa:



This screenshot shows the SAP ABAP Debugger interface in Standard mode, similar to the previous one but with a vertical dotted line on the left margin of the code editor, indicating the current execution point. The code content is identical to the previous screenshot.

A ferramenta Pilha ABAP e de Tela é um componente default da aba Standard, como dito anteriormente, serve para visualizarmos os pontos em que o programa já passou, permitindo voltar a esses pontos para a exibição dos valores, ou para a análise do código, vejamos esta ferramenta detalhadamente abaixo:

Po...	Nív....	T..	Tipo evento	Evento	Programa	Na...	Inclui
	2		PAI SCREEN	0101	SAPMV45A		
	1		TRANSACTION	VVA01(VA01)			

Indica em que bloco está o ponteiro

Nível do bloco

Nome do bloco

Nome do programa

Tipos de bloco:

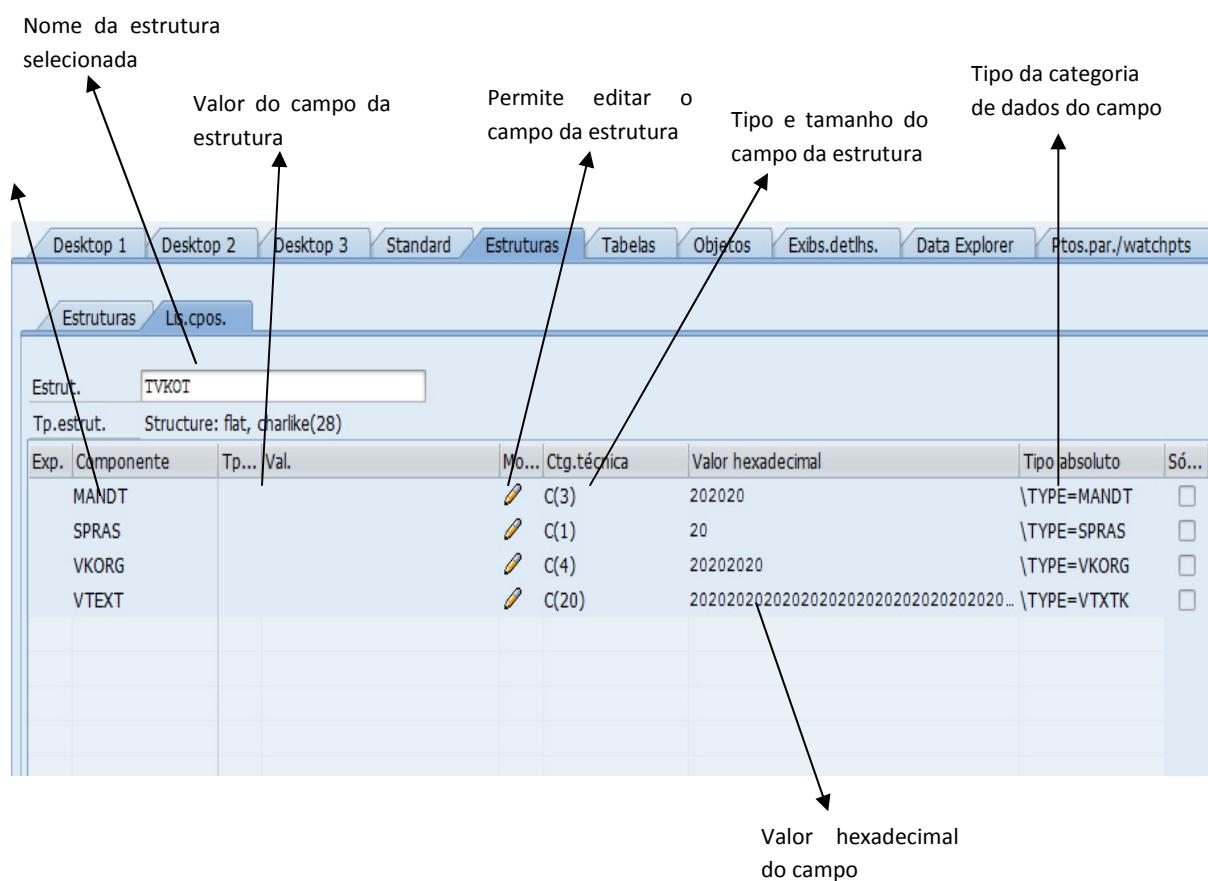
- PAI SCREEN
- TRANSACTION

Debugger

• **Aba Estruturas**



A aba estrutura mostra em uma visualização melhor as estruturas que você selecionar no programa, basta dar um duplo clique na estrutura e seleciona-la posteriormente que automaticamente esta ferramenta será exibida, permitindo realizar algumas operações, conforme veremos abaixo:



- **Aba Tabela**



A aba tabelas permite realizar operações específicas nas tabelas internas do programa, da mesma forma da opção anterior (Estruturas), porém mais completa, vejamos abaixo essas ferramentas detalhadamente:

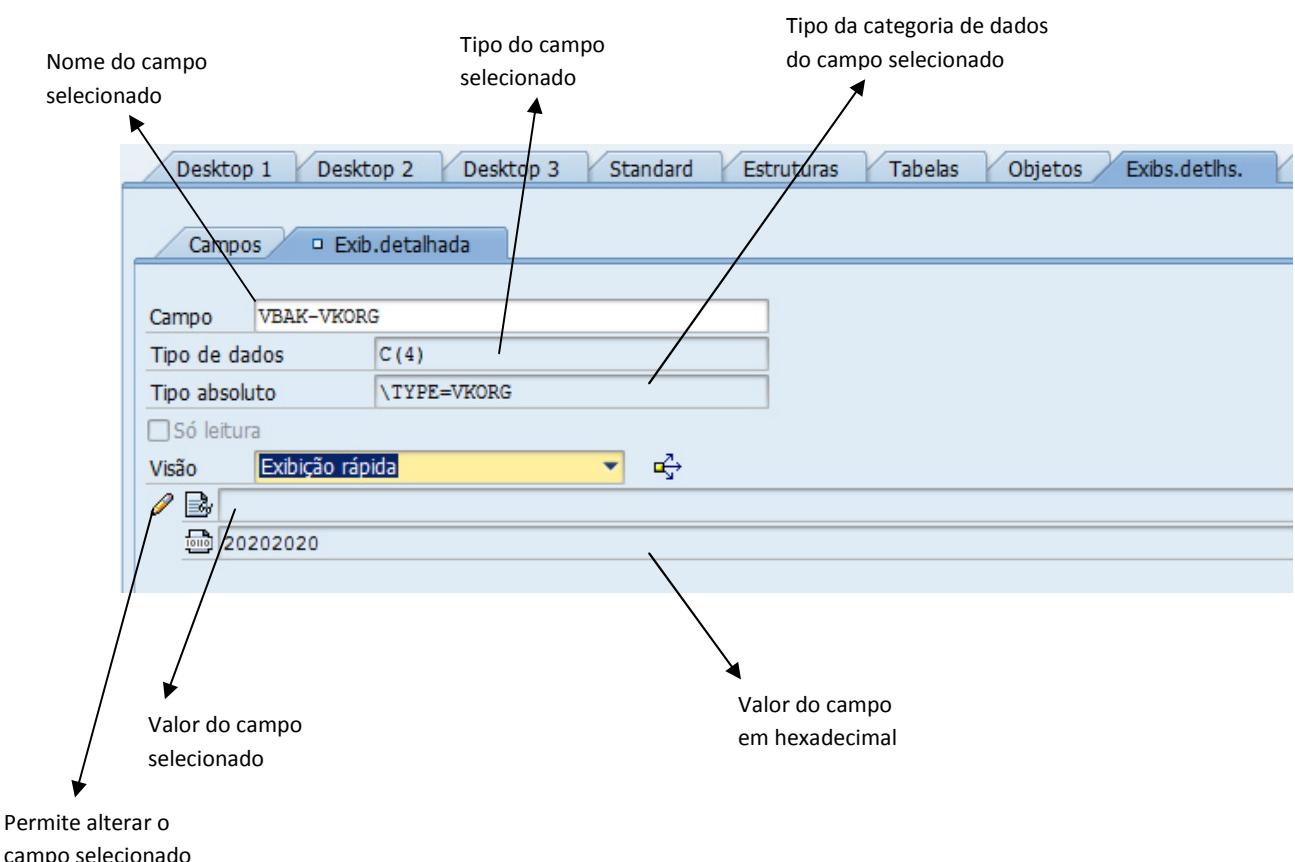
The screenshot shows the SAP Table View interface. The top navigation bar includes tabs for Desktop 1, Desktop 2, Desktop 3, Standard, Estruturas, Tabelas (selected), Objetos, and Exibs.detl. Below the tabs, there are two sub-tabs: Tabelas and Conteúdo tab.. The main area displays a table structure for the CVBRK table. The table has columns: Linha, MANDT..., VBELN..., FKART..., FKTYP..., VBTYP..., WAERK..., VKORG..., VTWEG..., KALSM..., and KNU. A toolbar above the table includes buttons for Inserir coluna, Colunas..., and other table operations. Annotations with arrows point to specific elements:

- Nome da tabela selecionada: Points to the table name field containing "CVBRK []".
- Permite inserir campos da tabela: Points to the "Inserir coluna" button.
- Layout das colunas exibidas: Points to the column header area.
- Exibe o cabeçalho da tabela: Points to the "Colunas..." button.
- Permite avançar numericamente as colunas: Points to the right arrow key in the numeric row selection field.
- Controle de visualização: Points to the rightmost arrow key in the numeric row selection field.
- Número da linha: Points to the numeric row selection field containing "1".
- Permite selecionar uma ou mais linhas: Points to the selection bar on the left side of the table.
- Campos da tabela, é possível editar os campos clicando duas vezes sobre o campo desejado.: Points to the table body where cells can be edited by double-clicking.

- **Aba Exibir Detalhadamente**



Toda vez que clicarmos duas vezes sobre um campo, essa ferramenta será automaticamente aberta, exibindo detalhes do campo e permitindo sua alteração, ela contém informações detalhadas de cada campo selecionado unicamente, vejamos abaixo algumas opções detalhadas dessa ferramenta.





- **Aba Pontos de parada/Watchpoints**

Permite visualizar, alterar, adicionar ou remover pontos de parada e Watchpoints, muito similar a ferramenta do debug antigo:

Dicas e Truques

O Debugger novo contém as mesmas dicas e truques do debug antigo, porém, uma se destaca principalmente no debug novo, muito utilizado para encontrar valores em programas para exits, badis, enhancements, vamos ver passo a passo como funciona essa dica que pode ser muito útil no dia a dia.

Quando estamos debugando um programa, principalmente os programas Standards, passamos por vários programas e alguns deles ficam para trás quando percebemos que precisamos de alguma variável que se encontra nele, porém, ao clicarmos duas vezes nessa variável, o ABAP irá mostrar que ela não existe mais, somente no momento em que ela estava em seu programa base, essa dica consiste em visualizar variáveis que estejam em outros pontos do programa, para que se possa recuperar valores, ver lógicas, etc.

Vejamos o exemplo abaixo:

O Editor passou por vários programas, conforme na imagem, temos a variável MATCONV, que está no programa SAPLOMCV, ao mudarmos de programa, essa variável irá perder os valores, pois não está mais em seu ponto inicial, vejamos a segunda imagem abaixo:

The screenshot shows the SAP ABAP Debugger interface. On the left is the code editor with ABAP code. In the center is the 'Pilha ABAP' (Call Stack) window, which lists the current stack frame (FUNCTION CONVERSION_EXIT_MAT...) at the top, followed by FORM CONVERSION_EXIT, EVENT SYSTEM-EXIT, and MODULE (PAI) VBSN_UNTERLEGEN. On the right is the 'Variáveis globais' (Global Variables) window, showing variables like TMCNV, WORK, MASKZE, MATCONV (with value MATCONV), and NUM (with value 1234567890).

Po...	Niv...	T..	Tipo evento	Evento	Programa	Na...	Inclui
4		FUNCTION	CONVERSION_EXIT_MA...	SAPLOMCV		LOMCV	
3		FORM	CONVERSION_EXIT	SAPCNVE		SAPCN	
2		EVENT	SYSTEM-EXIT	SAPMV45A		<SYST	
1		MODULE (PAI)	VBSN_UNTERLEGEN	SAPMV45A		MV45A	

T...	S...	Nome variável	C...	Val.	Tipo técnico
TMCNV				Structure: flat & not ch...	Flat Structure(68)
WORK					C(18)
MASKZE					C(18)
MATCONV				MATCONV	C(8)
NUM				1234567890	C(28)



Debugger

SAP

Conforme vemos, o programa agora está passando pelo ponto do programa principal SAPMV45A, neste programa não existe a variável MATCONV, por isso ela não está exibindo valores, mas se a necessidade se faz neste local, temos uma forma de ver o valor deste campo mesmo que ela tenha saído de seu programa principal, basta seguir a seguinte lógica abaixo da imagem:

The screenshot shows the SAP ABAP Debugger interface. On the left, the code editor displays a module named VBSN_UNTERLEGEN. The code includes several comments in German and some ABAP statements like PERFORM and CHECK. In the top navigation bar, the 'Standard' tab is selected. To the right of the code editor, the 'Pilha ABAP' (Stack) pane lists the execution stack with four entries: a function CONVERSION_EXIT_MA... (SAPLOMCV), a form CONVERSION_EXIT (SAPCNVE), an event SYSTEM-EXIT (SAPMV45A), and the current module (PAI) VBSN_UNTERLEGEN (SAPMV45A). Below the stack is the 'Variables 1' pane, which contains a table with columns S..., Variável, C..., Val., and M... Valor. A row for the variable 'MATCONV' is highlighted in yellow, showing its value as 4D4154434F4E5620.

E necessário usar () para recuperar o valor que está no outro programa, sempre seguindo da seguinte forma:

(Nome do programa principal) variável

Veja abaixo:

This screenshot shows the 'Variables 1' pane from the SAP ABAP Debugger. It has tabs for Variables 1, Variables 2, Locals, and Globals. The Variables 1 tab is active. The table displays a single entry for the variable '(SAPLOMCV) MATCONV' with the value 'MATCONV' in the 'Val.' column and the hexadecimal value '4D4154434F4E5620' in the 'Valor hexadecimal' column. There are icons for search, delete, and refresh at the top of the table.

Agora independente de que ponto que o programa esteja, você consegue visualizar os valores já deixados de lado anteriormente no processamento, essa lógica se aplica a todos os programas, só não funciona se o programa ainda não tiver passado por um ponto, mas basta seguir a Pilha ABAP para verificar quais os programas já foram utilizados.

Sidney Vidal

Consultant Sap Abap SR.

+55 11 4535-2405 (Escritório)

+55 11 97128-2414 (claro-whatsapp)

Skype: vidal1500

E-mail: vidal@mastersoft-ti.com.br

Site: <http://www.mastersoft-ti.com.br>