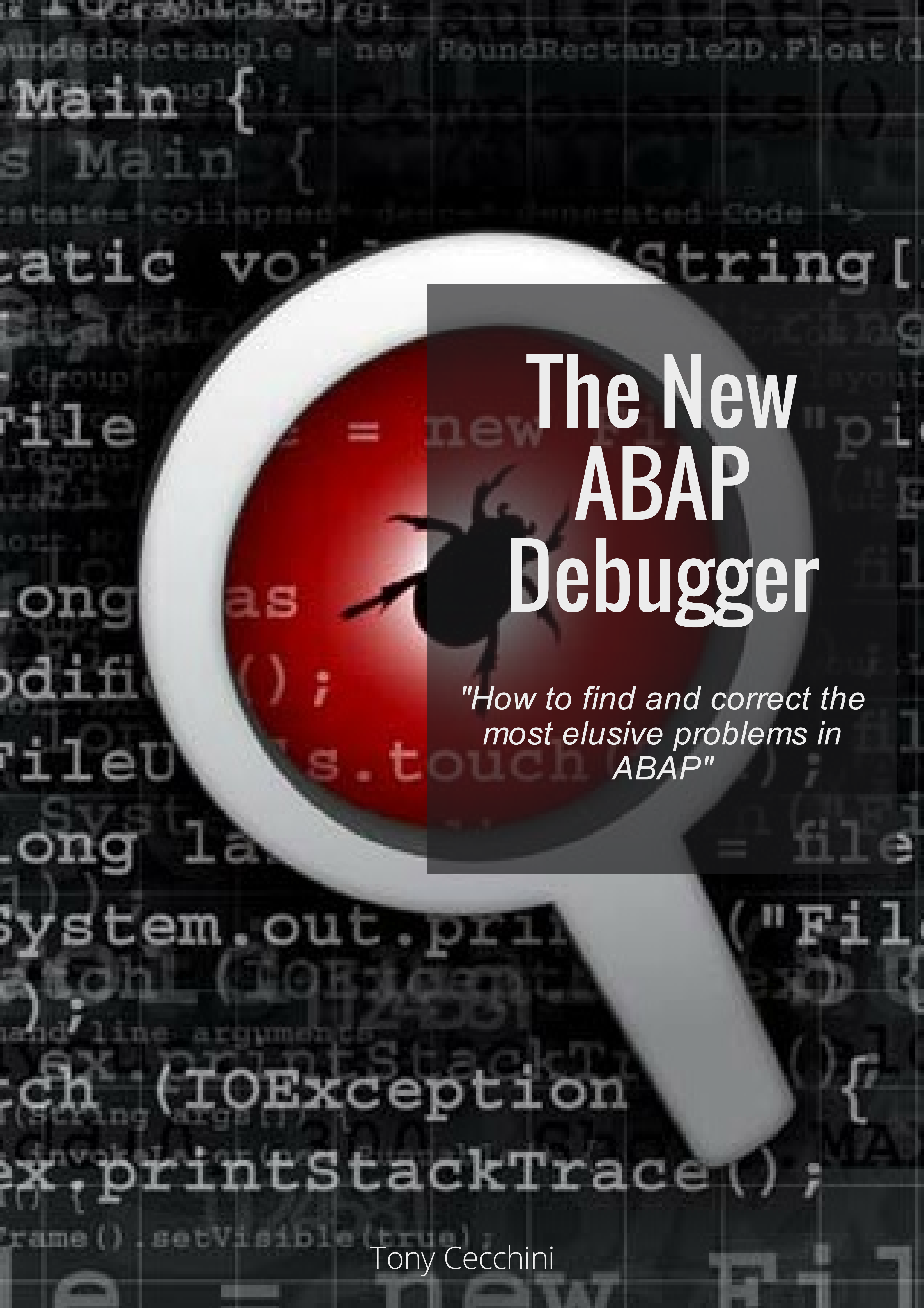


The New ABAP Debugger

"How to find and correct the most elusive problems in ABAP"

Tony Cecchini

This eBook will deal with the NEW ABAP debugger in ECC. Part 1 will explore the Classic (Old) debugger and the old architecture and drawbacks. It will then introduce the architecture for the new debugger and show how to make this your default-debugging tool for ECC. (It is the default Netweaver 2004s).

SAP introduced the "New ABAP Debugger" with SAP NetWeaver '04, which includes a new architecture that enables analysis of all ABAP programs and a state of the-art user interface. The architecture of the New ABAP Debugger, which has been developed from the ground up, will provide for a more flexible and intuitive user interface.If you happened to upgrade to SAP NetWeaver 2004s, then **New ABAP Debugger** has even more enhancements. Such as integration of the new ABAP Front-End Editor into the new source code display, complete with syntax highlighting and a Data Quick Info pop-up; the new Diff Tool, which guides you through the differences between, for example, two nested structures or internal tables; and the Data Explorer.

Why do we need a New ABAP Debugger?

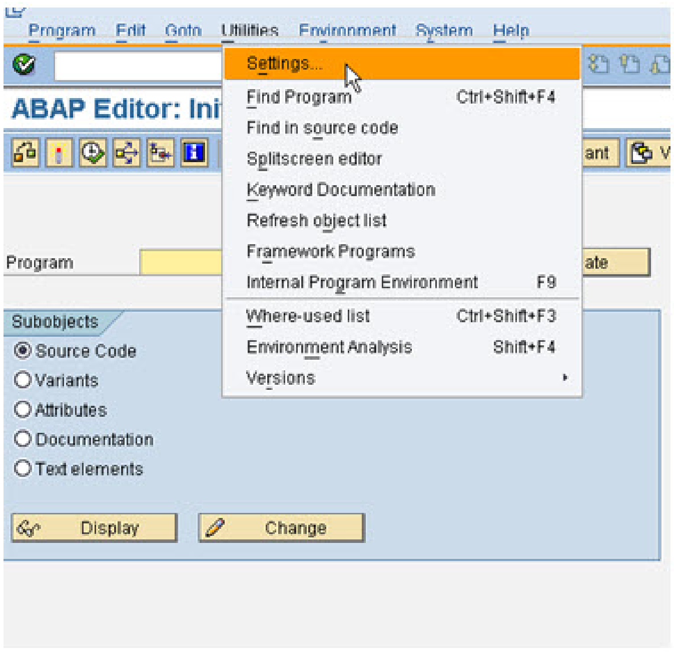
The Classic ABAP Debugger runs in the same roll area as the application to be analyzed (debuggee). It is therefore displayed in the same window as the application. However, this technology also has some restrictions. For instance, it is not possible to debug conversion exits and field exits. The reason behind this drawback is because inside a conversion exit for a field, no ABAP DIALOG statements are allowed, Like a CALL SCREEN. Since the debugger and the debguee share the same context, the debugger cannot send a debugger screen.

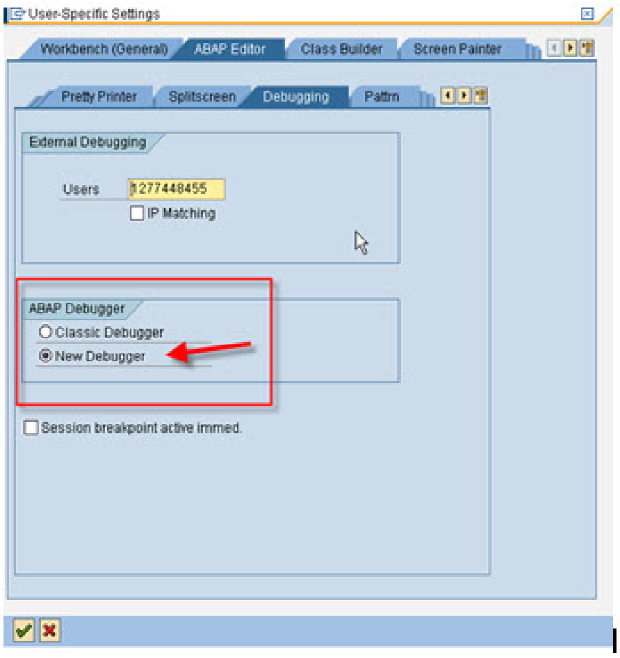
Another drawback is the risk of code in the debugger affecting the debugge. They both reside in the same internal session and naturally can impact each other. SAP has attempted to mitigate this risk but not using ABAP as the development language for the classic debuuger and its core resides in the Kernel. However, this leads to even more drawbacks, as now it is impossible to leverage any new ABAP GUI enhancements, such as the CFW (Control Framework).

New ABAP Debugger Architecture

Ok…now let us look at the new debugger architecture, called the Two-process architecture. So what is the big deal? The fundamental difference of this architecture is its two external sessions, each of which appears in its own SAP GUI window. One session for the debugge, and a separate session for the debugger. The debugger controls the debugge. This two-window approach, in which the debugger lives in its own external session, has significant advantages, and solved the issues of debugging through field exits and conversion routines. It has also insulated the object being debugged from the debugger instance. That means the integrity of debugge is no longer at risk of impact from the debugger.

OK, you want to use the new debugger…..Here is how to set it as your default. Switching between the Classic ABAP Debugger and the *New ABAP Debugger* is easy — in the Object Navigator (transaction SE80) or in the ABAP Editor (transaction SE38), follow the menu path Utilities–>Settings. (See Below)





Users 1277448455
[ ] IP Matching
ABAP Debugger: Classic Debugger [✓ New Debugger]
[ ] Session breakpoint active immed.

You can initiate the debugger as you already know how to do using /H or by setting a break-point.

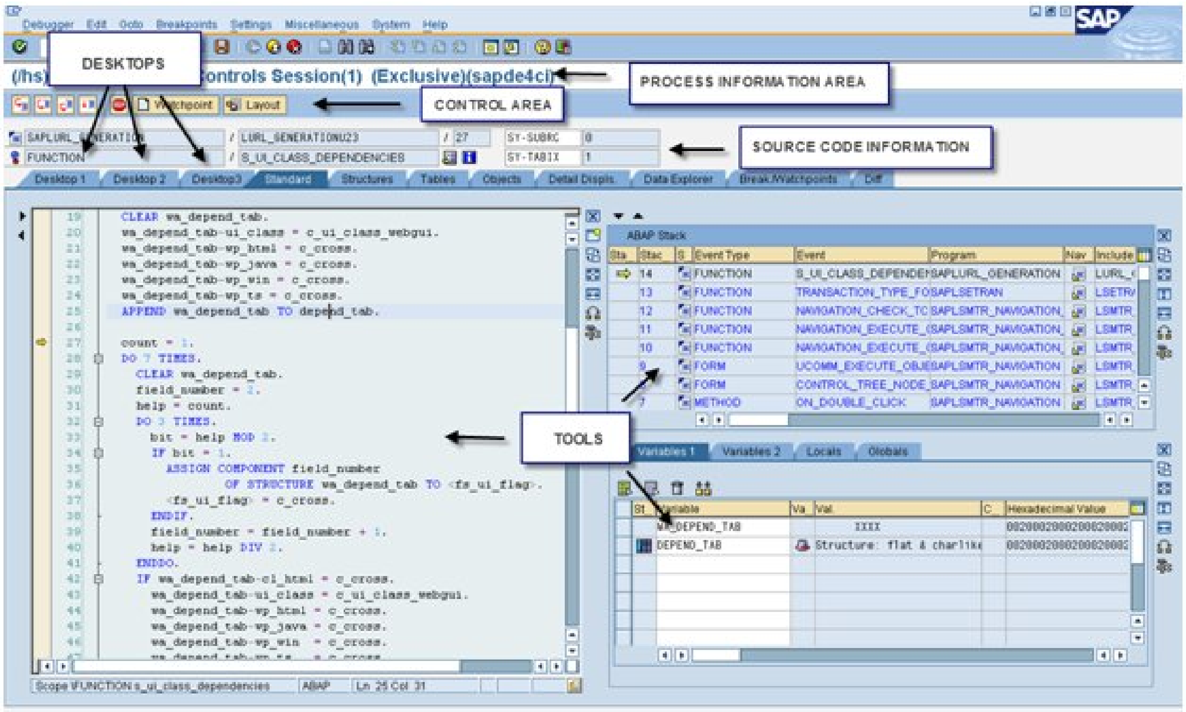
The New ABAP Debugger USER Interface

The **New ABAP Debugger** provides a state-of-the-art user interface that can be customized to your needs.

Before exploring how the *New ABAP Debugger* user interface can help you achieve greater productivity, let's focus on the following ......

- What are the main parts of the New ABAP Debugger user interface?
- How can I customize the ABAP Debugger User Interface?
- Which ABAP debugger tools are available through the user interface?

Let's look at the new ABAP Debugger interface. See below.



(/hs) ...ontrols Session(1) (Exclusive)(sapde4ci)

SAPLURL_GENERATION	/	LURL_GENERATIONU23	/ 27	SY-SUBRC	0
FUNCTION	/	S_UI_CLASS_DEPENDENCIES		SY-TABIX	1

```
CLEAR wa_depend_tab.
wa_depend_tab-ui_class = c_ui_class_webgui.
wa_depend_tab-wp_html = c_cross.
wa_depend_tab-wp_java = c_cross.
wa_depend_tab-wp_win = c_cross.
wa_depend_tab-wp_ts = c_cross.
APPEND wa_depend_tab TO depend_tab.

count = 1.
DO 7 TIMES.
  CLEAR wa_depend_tab.
  field_number = 2.
  help = count.
  DO 3 TIMES.
    bit = help MOD 2.
    IF bit = 1.
      ASSIGN COMPONENT field_number
              OF STRUCTURE wa_depend_tab TO <fs_ui_flag>.
      <fs_ui_flag> = c_cross.
    ENDIF.
    field_number = field_number + 1.
    help = help DIV 2.
  ENDDO.
  IF wa_depend_tab-cl_html = c_cross.
    wa_depend_tab-ui_class = c_ui_class_webgui.
    wa_depend_tab-wp_html = c_cross.
    wa_depend_tab-wp_java = c_cross.
    wa_depend_tab-wp_win  = c_cross.
```

Scope \FUNCTION s_ui_class_dependencies | ABAP | Ln 25 Col 31

Sta	Stac	S	Event Type	Event	Program	Nav	Include
⇨	14		FUNCTION	S_UI_CLASS_DEPENDE	SAPLURL_GENERATION		LURL_
	13		FUNCTION	TRANSACTION_TYPE_FO	SAPLSETRAN		LSETR
	12		FUNCTION	NAVIGATION_CHECK_TC	SAPLSMTR_NAVIGATION_		LSMTR_
	11		FUNCTION	NAVIGATION_EXECUTE_(	SAPLSMTR_NAVIGATION_		LSMTR_
	10		FUNCTION	NAVIGATION_EXECUTE_(	SAPLSMTR_NAVIGATION_		LSMTR_
	9		FORM	UCOMM_EXECUTE_OBJE	SAPLSMTR_NAVIGATION		LSMTR_
	8		FORM	CONTROL_TREE_NODE_	SAPLSMTR_NAVIGATION		LSMTR_
	7		METHOD	ON_DOUBLE_CLICK	SAPLSMTR_NAVIGATION		LSMTR_

St	Variable	Va	Val	C	Hexadecimal Value
	WA_DEPEND_TAB		XXXX		0020002000200020002
	DEPEND_TAB		Structure: flat & charlike		0020002000200020002

There are five main components of the :New ABAP Debugger user interface

- Process information area
- Control area
- Source code information area
- Desktops
- Tools

Let’s take a look at each in turn.

Process information area: The process information (or title) area of the ABAP debugger user interface provides information about the status of the debugger or debuggee. The following information is provided:

- **Session number:** Because you can debug several applications in parallel, you need to know which session the ABAP debugger user interface is connected to.

- **Debug setting/session type:** Next in the process information area is information about the session type or status:

(/hs) indicates that system debugging is active.

-HTTP- indicates HTTP debugging.

-RFC->destination indicates debugging of an RFC module at the specified destination.

-UPDATE- indicates debugging of the asynchronous update functionality of a transaction.

-ATTACHED- indicates that the debugger is attached to a process (you can attach the debugger to a running process via transaction SM50 by following the menu path Process–>Debugging.

- Exclusive/Non-Exclusive indicates whether the ABAP debugger session runs as an *exclusive* session or a *non-exclusive* session. In an exclusive debugging session, the work process is exclusively locked for your currently running debugging session; in a non-exclusive debugging session, the work process is not locked, and any debugger action may involve an implicit database commit during roll-in/roll-out of the debuggee context. (See July 2011 Q&A in Newsletter)

Control area: Standard features for execution control (step into, step over, return, continue) in the New ABAP Debugger are similar to those in the Classic ABAP Debugger. The new debugger also provides shortcuts to create breakpoints and watchpoints.

Source code information area: The New ABAP Debugger displays the full information about the current source position. The displayed information depends on the current code type — in ABAP code you see “main program,” “include,” and “source line” information; in screen flow code, you see “main program,” “screen number,” and “source line” information.

Desktops and tools: Finally, we reach the most important part of the New ABAP Debugger user interface — the desktops and tools. The desktops are your work areas, and all available tools (e.g.,the Source Code display tool and the Variable Fast Display and can be arranged on the desktops. You can configure the desktops to your needs and switch to specialized desktops for unique debugging tasks. As you can see, there are three user-specific desktops (Desktop 1, Desktop 2, and Desktop 3) and seven standard desktops (Standard, Structures, Tables, Objects, etc.).

You can customize which tools appear on any desktop (with a maximum of four tools on a desktop), along with the position and size of the tools on the desktop, but only the configuration of the three user-specific desktops can be stored permanently in a debugger variant. Thats right, we now have debugger variants.

A debugger variant can consist of Options (e.g., customizing the navigation to the Detail Views or special options for the different tools) and debugger settings (e.g., turn the system debugging on or off). You can store your Breakpoints and any user interface customizations (e.g., which tools are located on which desktop). Debugger variants are stored on the database. To access them, you need the name of user who created the variants and the variant name. You can also download a debugger variant to a local file. Cool…huh?

Customizing the New ABAP Debugger User Interface

Ok, let’s wrap this up with customizing the interface. Below you see a debugger tool (the Source Code display tool) residing on a desktop. It is automatically accompanied on the left by arrow symbols, which you can use to resize the tool, decreasing or increasing the size of the area it covers in the window. On the right side is an icon bar, which provides the following functionality:

- Close (or remove) the tool.

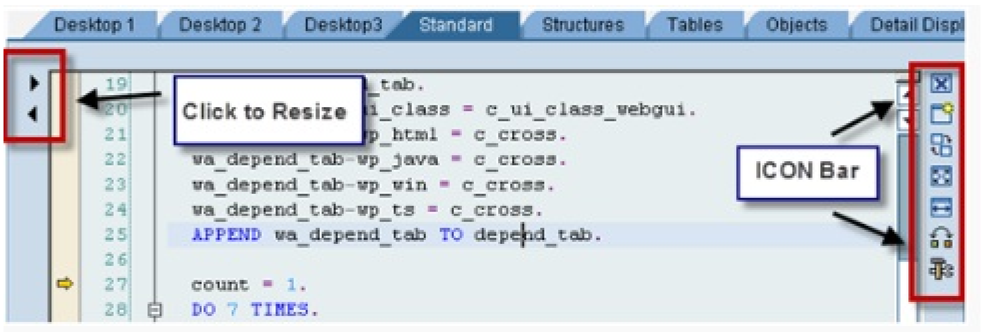
- Choose a tool from the pop-up dialog, and add it to the current desktop.

- Exchange the tool with another tool selected from the pop-up dialog.

- Set the size of the tool to full-screen mode (this hides all other tools – be careful)

- Swap the location of one tool with that of another tool.

- Invoke a tool-specific services dialog.



```
_tab.
i_class = c_ui_class_webgui.
p_html = c_cross.
wa_depend_tab-wp_java = c_cross.
wa_depend_tab-wp_win = c_cross.
wa_depend_tab-wp_ts = c_cross.
APPEND wa_depend_tab TO depend_tab.

count = 1.
DO 7 TIMES.
```

When you invoke a tool-specific services dialog, a services dialog pop up that is specific to the tool in use opens. Let's use the Tables tool as an example. (see below)

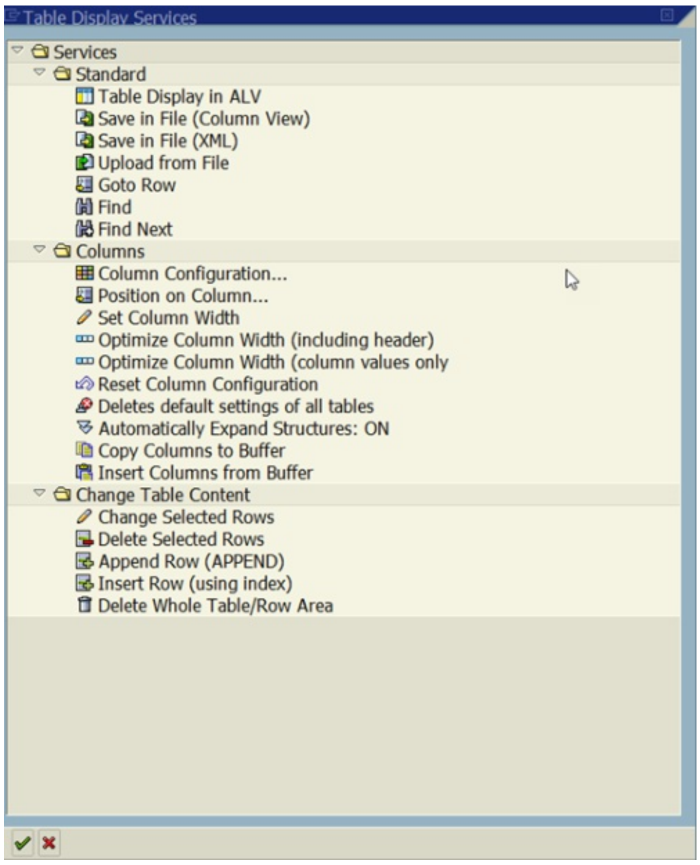


Table Display Services

- Services
  - Standard
    - Table Display in ALV
    - Save in File (Column View)
    - Save in File (XML)
    - Upload from File
    - Goto Row
    - Find
    - Find Next
  - Columns
    - Column Configuration...
    - Position on Column...
    - Set Column Width
    - Optimize Column Width (including header)
    - Optimize Column Width (column values only
    - Reset Column Configuration
    - Deletes default settings of all tables
    - Automatically Expand Structures: ON
    - Copy Columns to Buffer
    - Insert Columns from Buffer
  - Change Table Content
    - Change Selected Rows
    - Delete Selected Rows
    - Append Row (APPEND)
    - Insert Row (using index)
    - Delete Whole Table/Row Area

The services dialog consists of two sections:

Standard: Standard services are available for almost all tools. You can download the current tool content — for example, you can download the content of an internal table to a local file in order to analyze it by sorting the data, for example, in Microsoft Excel. You can also perform searches in the displayed content of the tool (press Ctrl-F to start a search and Ctrl-G to continue a search) — you could search for a special attribute of an object in the Object View.

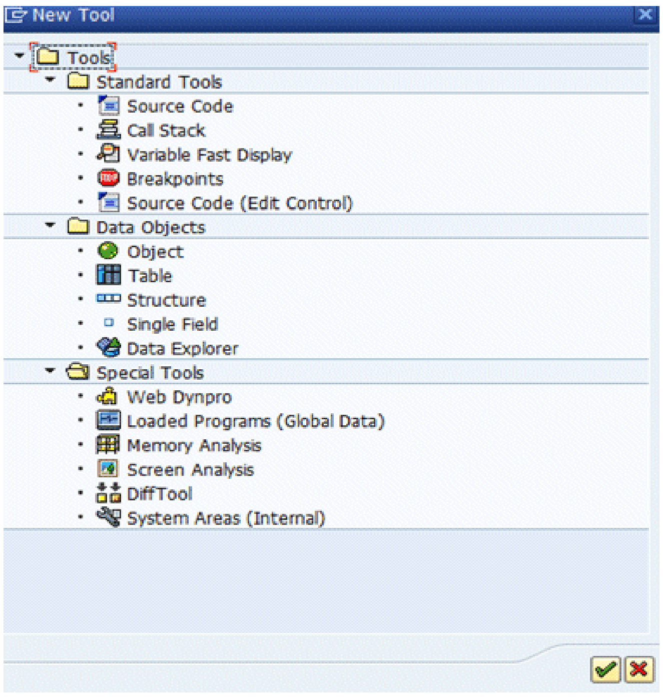
Tool-Specific: This portion of the dialog presents all the specific features of the selected tool. As shown in the table-specific services dialog above, there are tools for customizing the columns and for modifying the content of an internal table (e.g., changing or inserting a row). You even have the option to delete the table itself.

Tools of The NEW ABAP Debugger

In this, the 3rd and the final installment, I will explore the tools of the **NEW ABAP Debugger** somewhat in depth. I will explore the tools I feel you will most likely use repeatedly. Please consult the SAP help portal for the *New ABAP Debugger* to explore any tool further, and learn about any not discussed directly in this eBook.

The tools of the New ABAP Debugger are optimized for the different debugging situations you may have to deal with during the course of solving a problem. Some of the tools you will recognize from the Classic ABAP Debugger; others are brand new. Before we look at a few of the tools and how to use them, lets first get comfortable with the tools menu and how the tools are organized.

If you want to add a new tool to the current desktop, you will get the pop-up window titled "New Tool" where you can choose the appropriate tool based on your situation.


New Tool

- Tools
  - Standard Tools
    - Source Code
    - Call Stack
    - Variable Fast Display
    - Breakpoints
    - Source Code (Edit Control)
  - Data Objects
    - Object
    - Table
    - Structure
    - Single Field
    - Data Explorer
  - Special Tools
    - Web Dynpro
    - Loaded Programs (Global Data)
    - Memory Analysis
    - Screen Analysis
    - DiffTool
    - System Areas (Internal)

The New ABAP Debugger – Tool Sections

The tools are grouped into sections:

Standard Tools:

- **Source Code:** Display current source code in the **back-end editor**. Please note that this is the old editor and I recommend you do not use this. There is another tool farther down the menu that will use the NEW Editor

- Call Stack: Display the current ABAP stack and screen stacks.

- **Variable Fast Display:** Display variable value and type.

- **Breakpoints:** Maintain breakpoints, watchpoints, and checkpoints.

- **Source Code (Edit Control):** Display current source code in the **NEW ABAP Front-End Editor**. This is the option I recommend you use to enlist all the valuable enhancements of the new editor.

Data Objects:

- **Object:** Display/change objects and classes.

- **Table:** Display/change internal tables.

- **Structure:** Display/change structures.

- **Single Field:** Display/change variables with simple data types, such as C, N, D, T,STRING,... etc.

- **Data Explorer:** Display very complex data structures in a tree-like fashion.

Its worth mentioning, that if you double-click on a variable displayed in any tool, you will launch the corresponding detail view described above that corresponds to its data type.

Special Tools:

- **Web Dynpro:** The structure of the content of the current controller, the properties of the UI elements in the layout of the current view and the currently instantiated component usages

- **Loaded Programs (Global Data):** Display all currently loaded programs and their global variables.

- **Memory Analysis:** Display integrated Memory Inspector tool (S_MEMORY_INSPECTOR)

- **Screen Analysis:** Display the screen attributes and the subscreen stack.

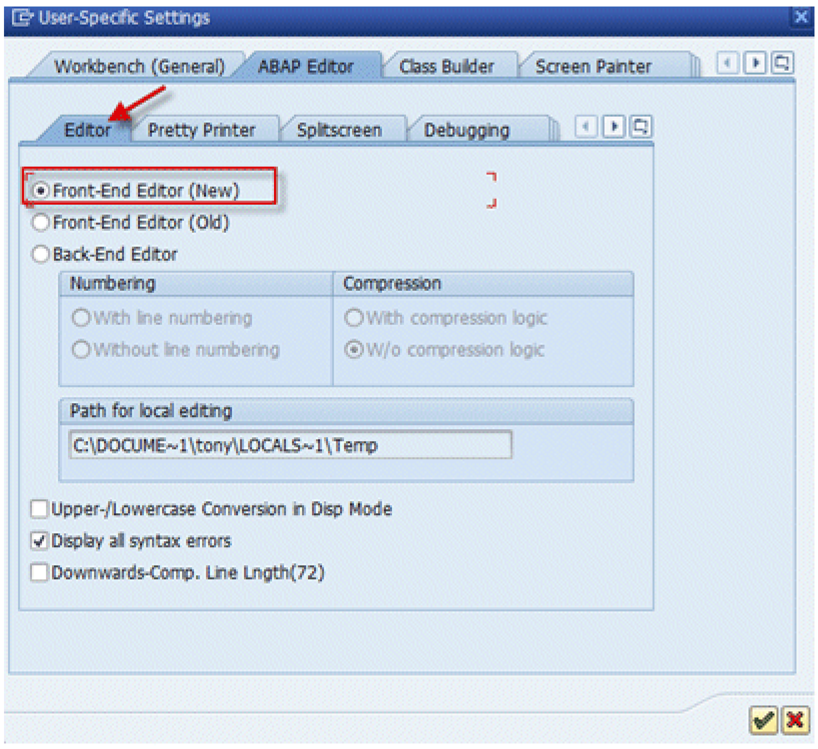
- **Diff Tool:** Compare variables very quickly to outline delta

- **System Areas (Internal):** Display system areas such as SMEM, ABAP Memory, Datasets, Screen Attributes...etc

OK, lets take a closer look at the most common debugging activities — displaying source code and the call stack, analyzing variables, and setting breakpoints and watchpoints.

Displaying ABAP Source code

I believe it's important that we are using the New Editor and not the old "Classic Editor". To be sure you are using the new editor, from the ABAP Editor main screen (SE38 or SE80) follow the menu path Utilities–>Settings. Verify that you are positioned on the editor tab and have selected the radio button for the Front-End Editor (New).



(●) Front-End Editor (New)
( ) Front-End Editor (Old)
( ) Back-End Editor

Numbering
( ) With line numbering
( ) Without line numbering

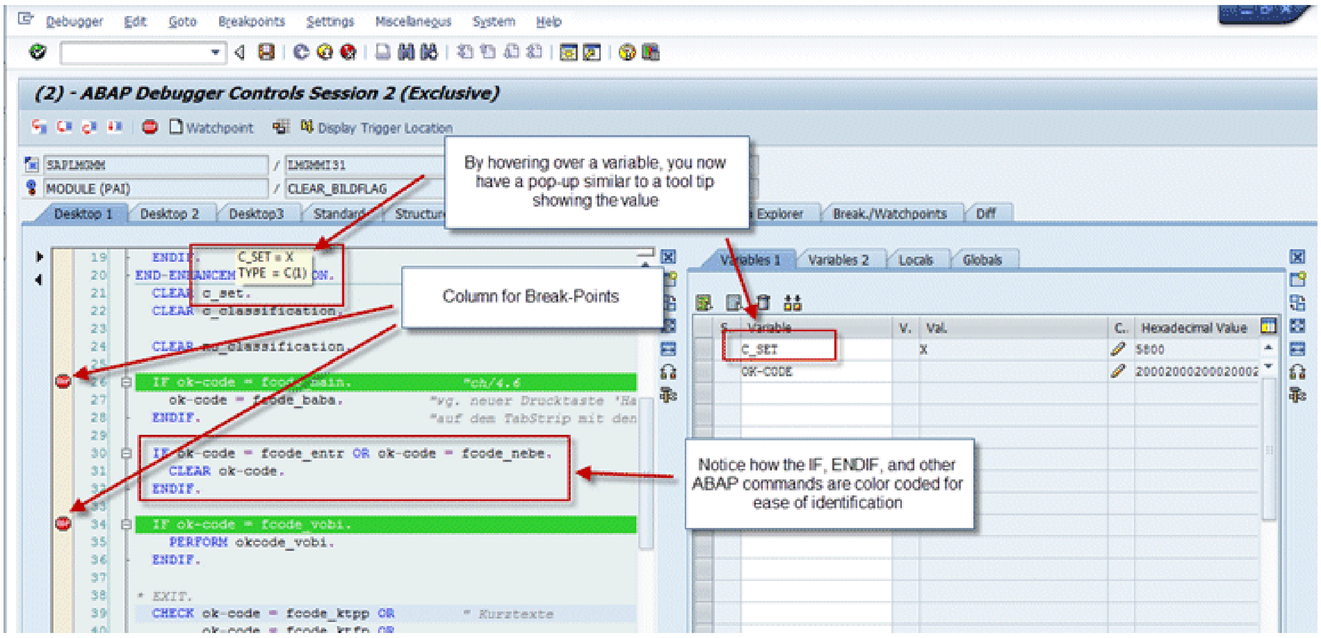
Compression
( ) With compression logic
(●) W/o compression logic

Path for local editing: C:\DOCUME~1\tony\LOCALS~1\Temp

☐ Upper-/Lowercase Conversion in Disp Mode
☑ Display all syntax errors
☐ Downwards-Comp. Line Lngth(72)

There are several compelling reasons for doing this (look for an upcoming Blog on the New SAP Editor), the reason most relevant for this blog is that the New ABAP Debugger leverages the following benefits of the new ABAP Front-End Editor…

-ABAP code is easy to read because of real-time syntax coloring.
–Both vertical and horizontal free scrolling is possible.
–Processing block start and end (i.e., IF/ENDIF,LOOP/ENDLOOP) is highlighted in real time.
–Variable values and types appear in a Data Quick Info pop-up.
–Breakpoints are easily set using a breakpoint column



Debugger Edit Goto Breakpoints Settings Miscellaneous System Help
Watchpoint
Display Trigger Location
SAPLMGMM / LMGMMI31
MODULE (PAI) / CLEAR_BILDFLAG
Desktop 1 Desktop 2 Desktop3 Standard Structure Explorer Break./Watchpoints Diff
Variables 1 Variables 2 Locals Globals

(2) - ABAP Debugger Controls Session 2 (Exclusive)

By hovering over a variable, you now have a pop-up similar to a tool tip showing the value

C_SET = X
TYPE = C(1)

Column for Break-Points

Notice how the IF, ENDIF, and other ABAP commands are color coded for ease of identification

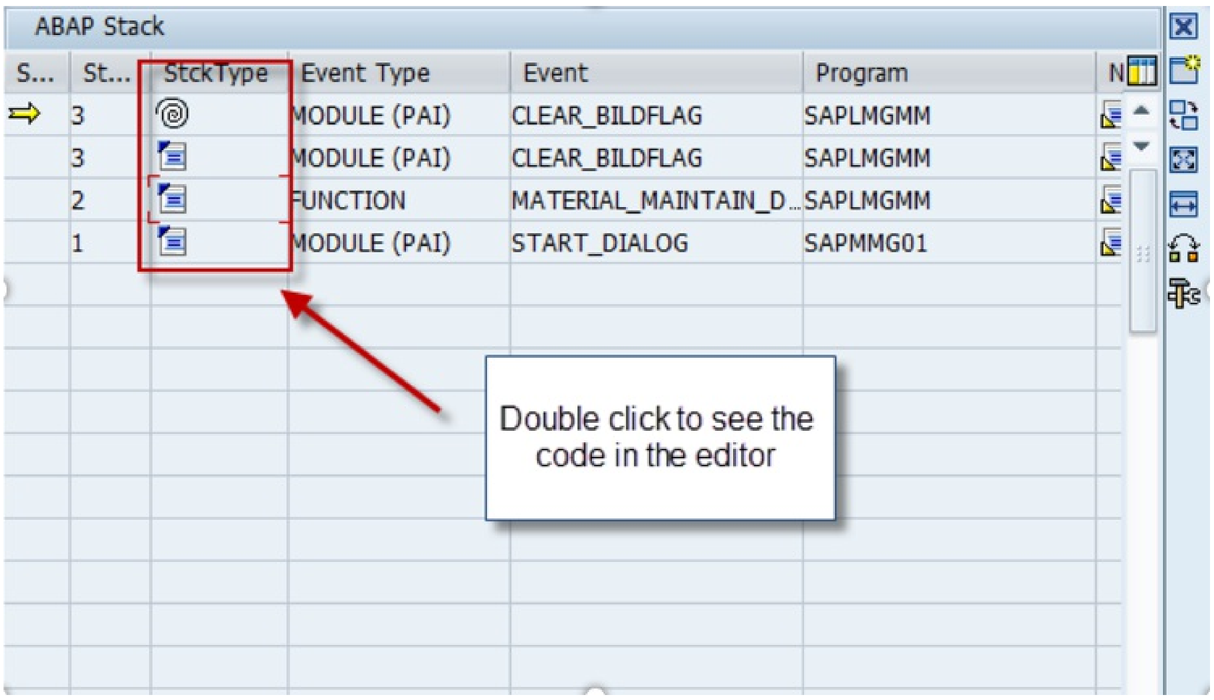
```
  ENDIF.
END-ENHANCEMENT-SECTION.
  CLEAR c_set.
  CLEAR c_classification.

  CLEAR no_classification.

  IF ok-code = fcode_main.                    "ch/4.6
    ok-code = fcode_baba.                     "vg. neuer Drucktaste 'Ha
  ENDIF.                                      "auf dem TabStrip mit den

  IF ok-code = fcode_entr OR ok-code = fcode_nebe.
    CLEAR ok-code.
  ENDIF.

  IF ok-code = fcode_vobi.
    PERFORM okcode_vobi.
  ENDIF.

* EXIT.
  CHECK ok-code = fcode_ktpp OR               " Kurztexte
        ok-code = fcode_ktfn OR
```

S.	Variable	V.	Val.	C.	Hexadecimal Value
	C_SET		X		5800
	OK-CODE				20002000200020002

Displaying the ABAP CALL Stack in The New ABAP Debugger

The Call Stack tool displays the ABAP call stack. It allows you to switch to the context of each stack level and navigate to the source code by opening the ABAP Editor, in order to start a deeper analysis of the code in a separate window(just double click!). Also new with SAP NetWeaver 2004s is the ability to display the screen (Dynpro) stack as well. Even more important, you can display a combined ABAP and screen stack to find out on which ABAP level which screen was called, and which screen invoked which ABAP module.

You can navigate from each stack line to the corresponding source line in the editor by double clicking on the icon in the stack type column.



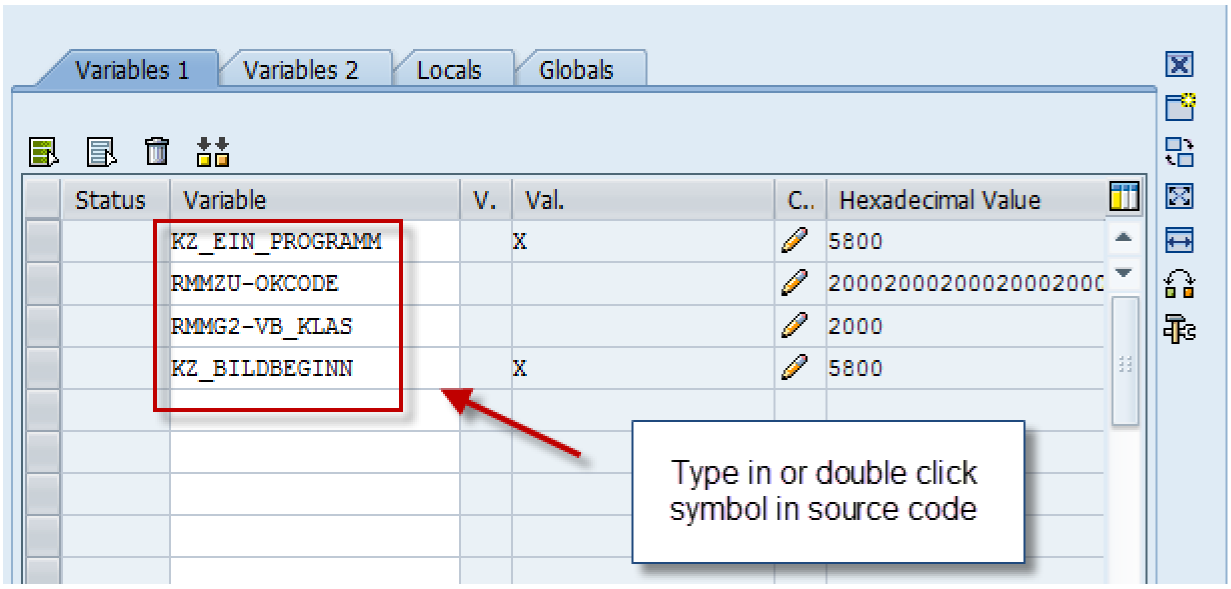
ABAP Stack

S...	St...	StckType	Event Type	Event	Program
⇨	3		MODULE (PAI)	CLEAR_BILDFLAG	SAPLMGMM
	3		MODULE (PAI)	CLEAR_BILDFLAG	SAPLMGMM
	2		FUNCTION	MATERIAL_MAINTAIN_D...	SAPLMGMM
	1		MODULE (PAI)	START_DIALOG	SAPMMG01

Displaying variable contents

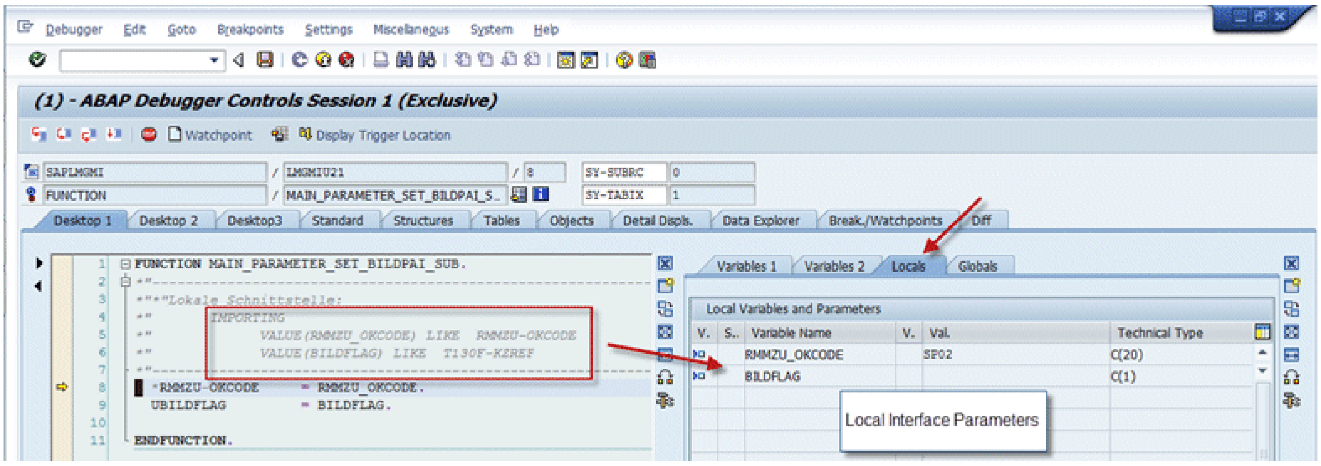
One of the new features of the New ABAP Debugger is the Variable Fast Display tool, which enables you to view basic information about a variable, such as the variable type, value, hexadecimal value, etc.

To display this information, you select the tool and then type in the variable name, or you can double-click on the variable name (i.e., the symbol) in the source code to transfer the symbol to the Variable Fast Display.



Status	Variable	V.	Val.	C..	Hexadecimal Value
	KZ_EIN_PROGRAMM		X		5800
	RMMZU-OKCODE				2000200020002000200C
	RMMG2-VB_KLAS				2000
	KZ_BILDBEGINN		X		5800

Even more convenient is the ability to display all global variables of the current program or all local variables (including the interface parameters) of the current procedure.



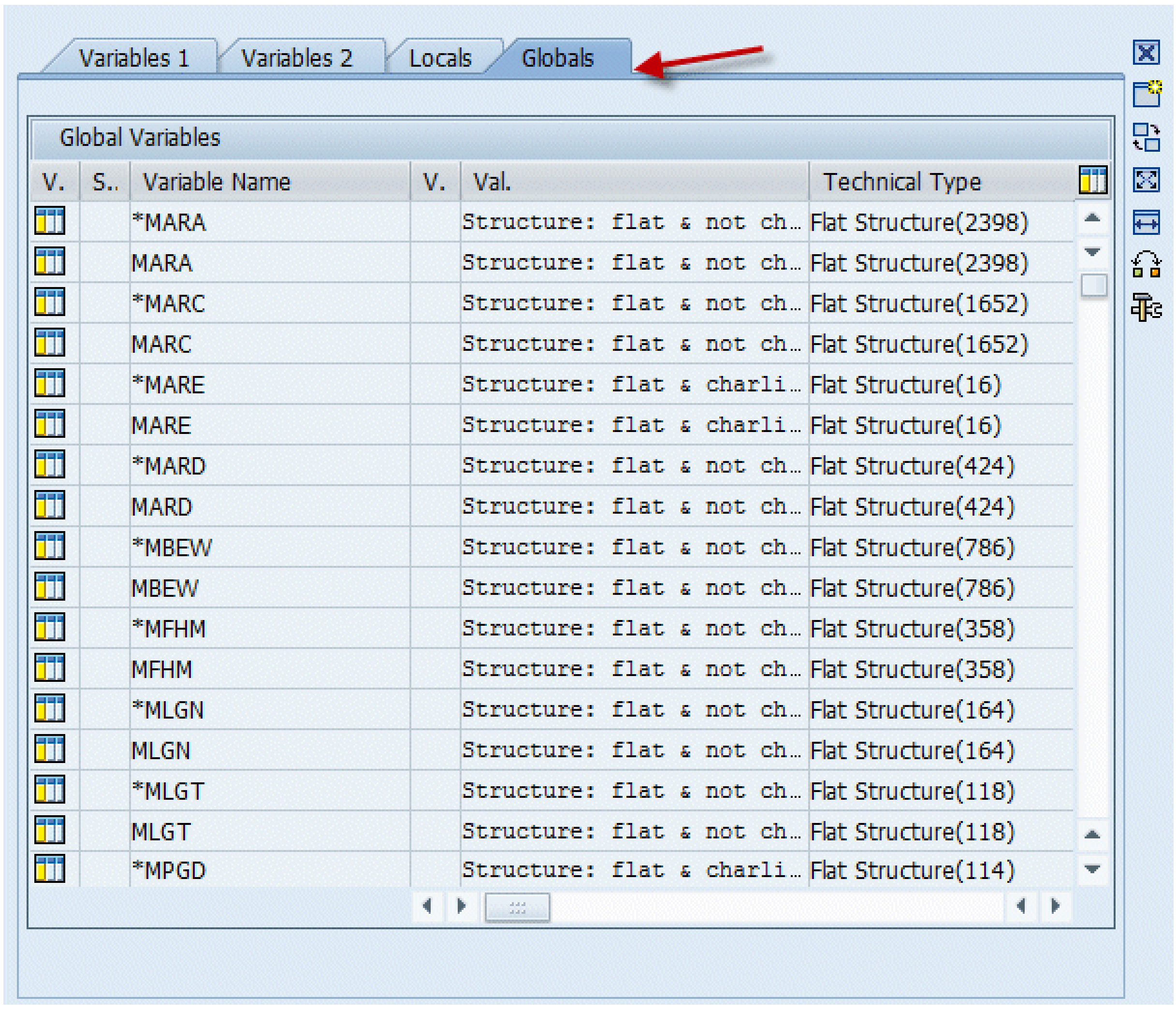
(1) - ABAP Debugger Controls Session 1 (Exclusive)

SAPLMGMI	/	LMGMIU21	/ 8	SY-SUBRC	0
FUNCTION	/	MAIN_PARAMETER_SET_BILDPAI_S...		SY-TABIX	1

```
FUNCTION MAIN_PARAMETER_SET_BILDPAI_SUB.
*"----------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(RMMZU_OKCODE) LIKE  RMMZU-OKCODE
*"             VALUE(BILDFLAG) LIKE  T130F-KZREF
*"----------------------------------------------------------------
  *RMMZU-OKCODE        = RMMZU_OKCODE.
  UBILDFLAG           = BILDFLAG.

ENDFUNCTION.
```

Local Variables and Parameters

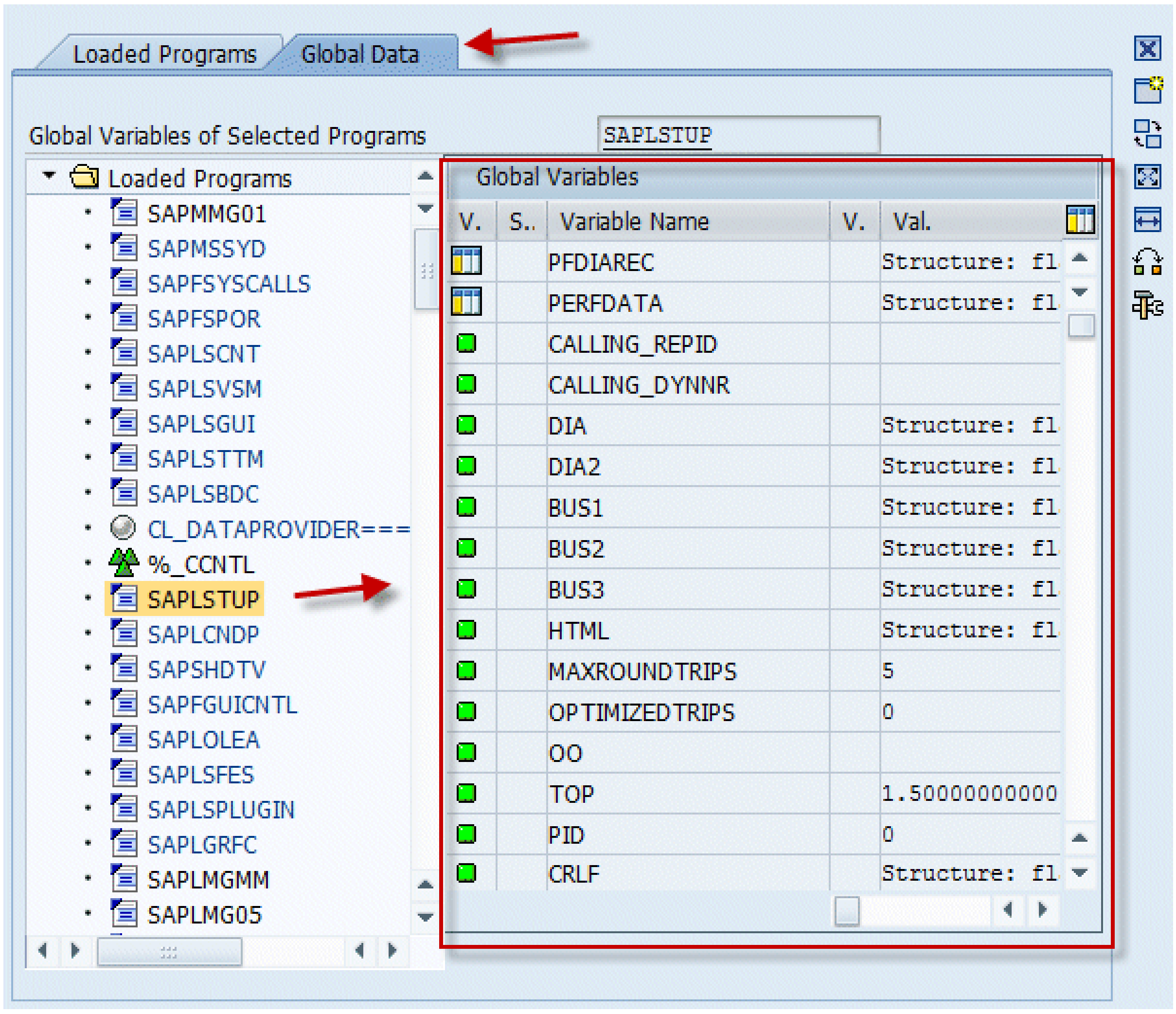
V.	S..	Variable Name	V.	Val.	Technical Type
		RMMZU_OKCODE		SP02	C(20)
		BILDFLAG			C(1)



Global Variables

V.	S..	Variable Name	V.	Val.	Technical Type
▦		*MARA		Structure: flat & not ch…	Flat Structure(2398)
▦		MARA		Structure: flat & not ch…	Flat Structure(2398)
▦		*MARC		Structure: flat & not ch…	Flat Structure(1652)
▦		MARC		Structure: flat & not ch…	Flat Structure(1652)
▦		*MARE		Structure: flat & charli…	Flat Structure(16)
▦		MARE		Structure: flat & charli…	Flat Structure(16)
▦		*MARD		Structure: flat & not ch…	Flat Structure(424)
▦		MARD		Structure: flat & not ch…	Flat Structure(424)
▦		*MBEW		Structure: flat & not ch…	Flat Structure(786)
▦		MBEW		Structure: flat & not ch…	Flat Structure(786)
▦		*MFHM		Structure: flat & not ch…	Flat Structure(358)
▦		MFHM		Structure: flat & not ch…	Flat Structure(358)
▦		*MLGN		Structure: flat & not ch…	Flat Structure(164)
▦		MLGN		Structure: flat & not ch…	Flat Structure(164)
▦		*MLGT		Structure: flat & not ch…	Flat Structure(118)
▦		MLGT		Structure: flat & not ch…	Flat Structure(118)
▦		*MPGD		Structure: flat & charli…	Flat Structure(114)

If you are interested in a special global variable of another program that is part of the application, you can use the Loaded Programs (Global Data) tool belonging to the special tools category, which allows you to browse all global variables of all loaded programs, not just the current program.



Loaded Programs | Global Data

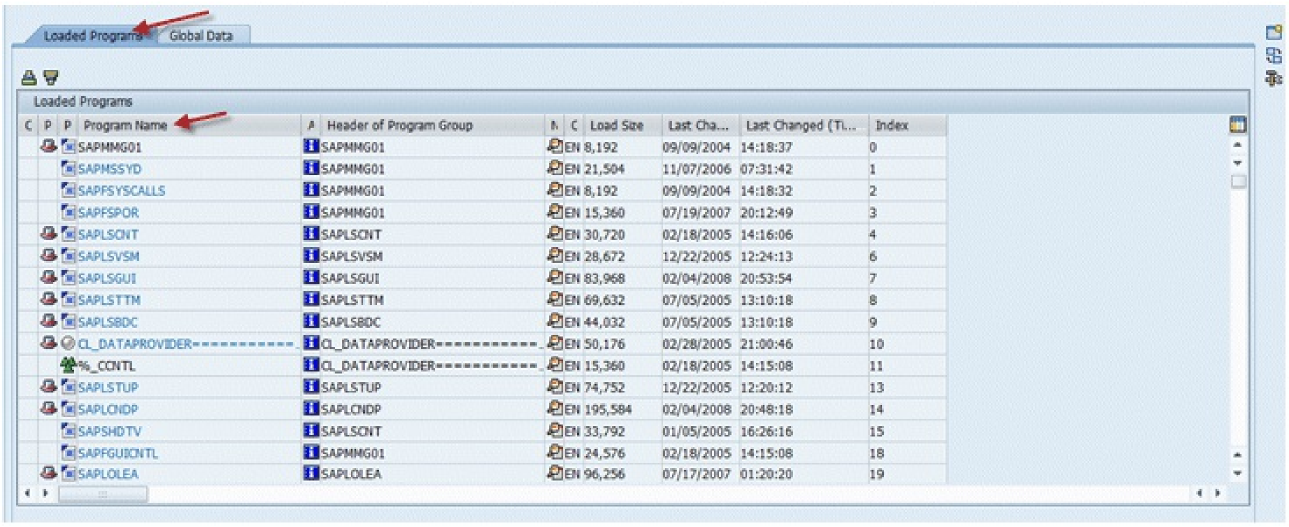
Global Variables of Selected Programs: SAPLSTUP

- Loaded Programs
  - SAPMMG01
  - SAPMSSYD
  - SAPFSYSCALLS
  - SAPFSPOR
  - SAPLSCNT
  - SAPLSVSM
  - SAPLSGUI
  - SAPLSTTM
  - SAPLSBDC
  - CL_DATAPROVIDER===
  - %_CCNTL
  - SAPLSTUP
  - SAPLCNDP
  - SAPSHDTV
  - SAPFGUICNTL
  - SAPLOLEA
  - SAPLSFES
  - SAPLSPLUGIN
  - SAPLGRFC
  - SAPLMGMM
  - SAPLMG05

Global Variables

V.	S..	Variable Name	V.	Val.
		PFDIAREC		Structure: fl
		PERFDATA		Structure: fl
🟩		CALLING_REPID		
🟩		CALLING_DYNNR		
🟩		DIA		Structure: fl
🟩		DIA2		Structure: fl
🟩		BUS1		Structure: fl
🟩		BUS2		Structure: fl
🟩		BUS3		Structure: fl
🟩		HTML		Structure: fl
🟩		MAXROUNDTRIPS		5
🟩		OPTIMIZEDTRIPS		0
🟩		OO		
🟩		TOP		1.50000000000
🟩		PID		0
🟩		CRLF		Structure: fl

This tool also allows you to check out which programs have been used (i.e., loaded) in the current internal session.



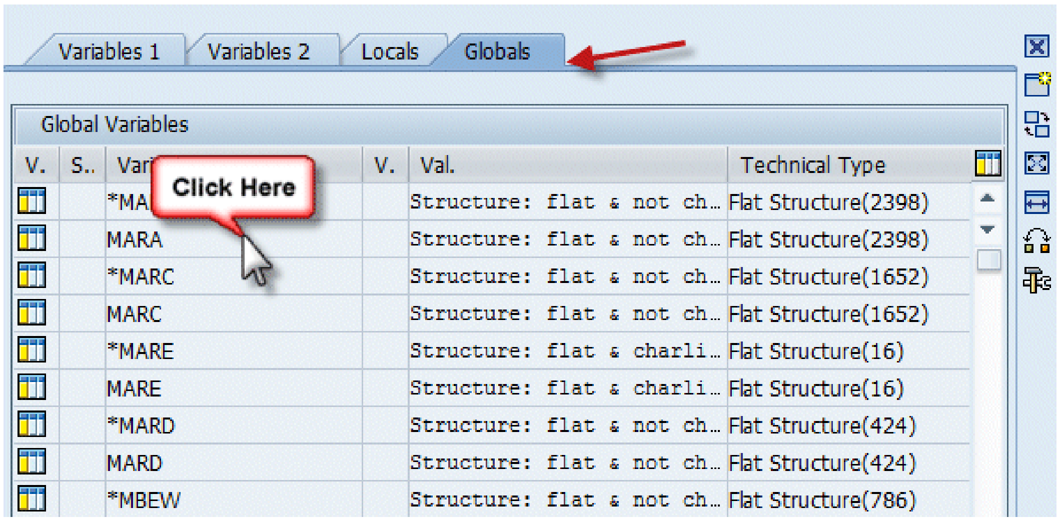
Loaded Programs | Global Data

Loaded Programs

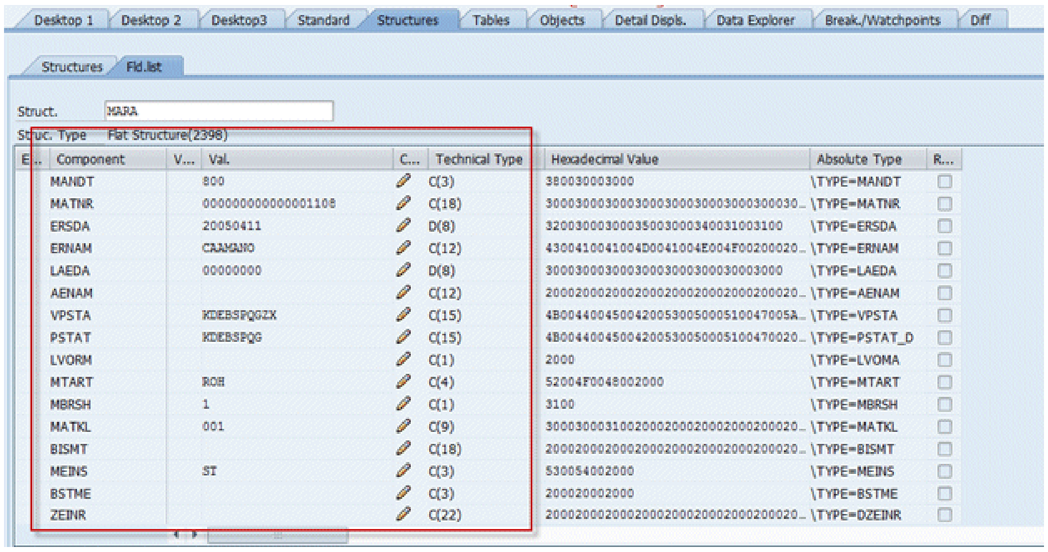
C	P	P	Program Name	A	Header of Program Group	N	C	Load Size	Last Cha...	Last Changed (Ti...	Index
			SAPMMG01		SAPMMG01		EN	8,192	09/09/2004	14:18:37	0
			SAPMSSYD		SAPMMG01		EN	21,504	11/07/2006	07:31:42	1
			SAPFSYSCALLS		SAPMMG01		EN	8,192	09/09/2004	14:18:32	2
			SAPFSPOR		SAPMMG01		EN	15,360	07/19/2007	20:12:49	3
			SAPLSCNT		SAPLSCNT		EN	30,720	02/18/2005	14:16:06	4
			SAPLSVSM		SAPLSVSM		EN	28,672	12/22/2005	12:24:13	6
			SAPLSGUI		SAPLSGUI		EN	83,968	02/04/2008	20:53:54	7
			SAPLSTTM		SAPLSTTM		EN	69,632	07/05/2005	13:10:18	8
			SAPLSBDC		SAPLSBDC		EN	44,032	07/05/2005	13:10:18	9
			CL_DATAPROVIDER===========_		CL_DATAPROVIDER===========_		EN	50,176	02/28/2005	21:00:46	10
			%_CCNTL		CL_DATAPROVIDER===========_		EN	15,360	02/18/2005	14:15:08	11
			SAPLSTUP		SAPLSTUP		EN	74,752	12/22/2005	12:20:12	13
			SAPLCNDP		SAPLCNDP		EN	195,584	02/04/2008	20:48:18	14
			SAPSHDTV		SAPLSCNT		EN	33,792	01/05/2005	16:26:16	15
			SAPFGUICNTL		SAPMMG01		EN	24,576	02/18/2005	14:15:08	18
			SAPLOLEA		SAPLOLEA		EN	96,256	07/17/2007	01:20:20	19

One of the most significant benefits of the New ABAP Debugger is the ability to provide tailored detail views for all ABAP data types. The detail views are part of the New ABAP Debugger navigation system. Regardless of the tool that is active, you can navigate to a detail view by simply double-clicking on the variable in any tool (with the exception of the Source Code tool), and the variable automatically appears in the appropriate detail view.

For example, if we go back to the above diagram for global variables for the current program and double-click on line 2 of the MARA structure, the tool displays the Structure of MARA at the discrete field level.



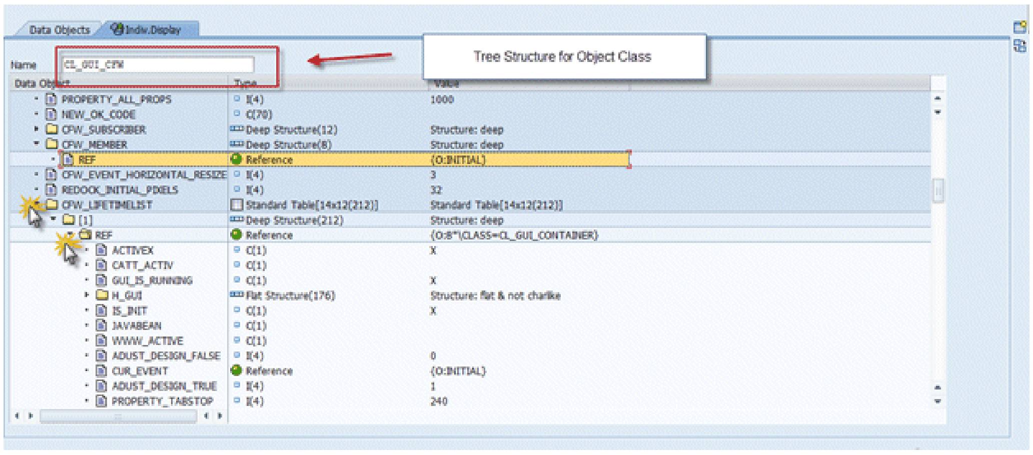
V.	S..	Var...	V.	Val.	Technical Type
▦		*MA...		Structure: flat & not ch…	Flat Structure(2398)
▦		MARA		Structure: flat & not ch…	Flat Structure(2398)
▦		*MARC		Structure: flat & not ch…	Flat Structure(1652)
▦		MARC		Structure: flat & not ch…	Flat Structure(1652)
▦		*MARE		Structure: flat & charli…	Flat Structure(16)
▦		MARE		Structure: flat & charli…	Flat Structure(16)
▦		*MARD		Structure: flat & not ch…	Flat Structure(424)
▦		MARD		Structure: flat & not ch…	Flat Structure(424)
▦		*MBEW		Structure: flat & not ch…	Flat Structure(786)



Struct. MARA
Struc. Type Flat Structure(2398)

E...	Component	V...	Val.	C...	Technical Type	Hexadecimal Value	Absolute Type	R...
	MANDT		800	✎	C(3)	380030003000	\TYPE=MANDT	☐
	MATNR		000000000000001108	✎	C(18)	30003000300030003000300030003000300030…	\TYPE=MATNR	☐
	ERSDA		20050411	✎	D(8)	32003000300035003000340031003100	\TYPE=ERSDA	☐
	ERNAM		CAAMANO	✎	C(12)	4300410041004D0041004E004F00200020…	\TYPE=ERNAM	☐
	LAEDA		00000000	✎	D(8)	30003000300030003000300030003000	\TYPE=LAEDA	☐
	AENAM			✎	C(12)	20002000200020002000200020002000200020…	\TYPE=AENAM	☐
	VPSTA		KDEBSPQGZX	✎	C(15)	4B0044004500420053005000510047005A…	\TYPE=VPSTA	☐
	PSTAT		KDEBSPQG	✎	C(15)	4B00440045004200530050005100470020…	\TYPE=PSTAT_D	☐
	LVORM			✎	C(1)	2000	\TYPE=LVOMA	☐
	MTART		ROH	✎	C(4)	52004F0048002000	\TYPE=MTART	☐
	MBRSH		1	✎	C(1)	3100	\TYPE=MBRSH	☐
	MATKL		001	✎	C(9)	30003000310020002000200020002000200020…	\TYPE=MATKL	☐
	BISMT			✎	C(18)	20002000200020002000200020002000200020…	\TYPE=BISMT	☐
	MEINS		ST	✎	C(3)	530054002000	\TYPE=MEINS	☐
	BSTME			✎	C(3)	200020002000	\TYPE=BSTME	☐
	ZEINR			✎	C(22)	20002000200020002000200020002000200020…	\TYPE=DZEINR	☐

For very complicated data structures, such as a complex nested object, it is more convenient to start an analysis in a tree-like display that allows you to dig deeper and deeper into the object without losing information about the levels above. The Data Explorer tool provides this capability.



Name CL_GUI_CFW

Data Object	Type	Value
PROPERTY_ALL_PROPS	I(4)	1000
NEW_OK_CODE	C(70)	
CFW_SUBSCRIBER	Deep Structure(12)	Structure: deep
CFW_MEMBER	Deep Structure(8)	Structure: deep
REF	Reference	{O:INITIAL}
CFW_EVENT_HORIZONTAL_RESIZE	I(4)	3
REDOCK_INITIAL_PIXELS	I(4)	32
CFW_LIFETIMELIST	Standard Table[14x12(212)]	Standard Table[14x12(212)]
[1]	Deep Structure(212)	Structure: deep
REF	Reference	{O:8*\CLASS=CL_GUI_CONTAINER}
ACTIVEX	C(1)	X
CATT_ACTIV	C(1)	
GUI_IS_RUNNING	C(1)	X
H_GUI	Flat Structure(176)	Structure: flat & not charlike
IS_INIT	C(1)	X
JAVABEAN	C(1)	
WWW_ACTIVE	C(1)	
ADJUST_DESIGN_FALSE	I(4)	0
CUR_EVENT	Reference	{O:INITIAL}
ADJUST_DESIGN_TRUE	I(4)	1
PROPERTY_TABSTOP	I(4)	240

Break-Points and Watch-Points in The New ABAP Debugger

While breakpoints have not changed much between the Classic and New ABAP Debuggers,save a few enhancements, watchpoints have. Let's take a look at each in turn.

Break-Points

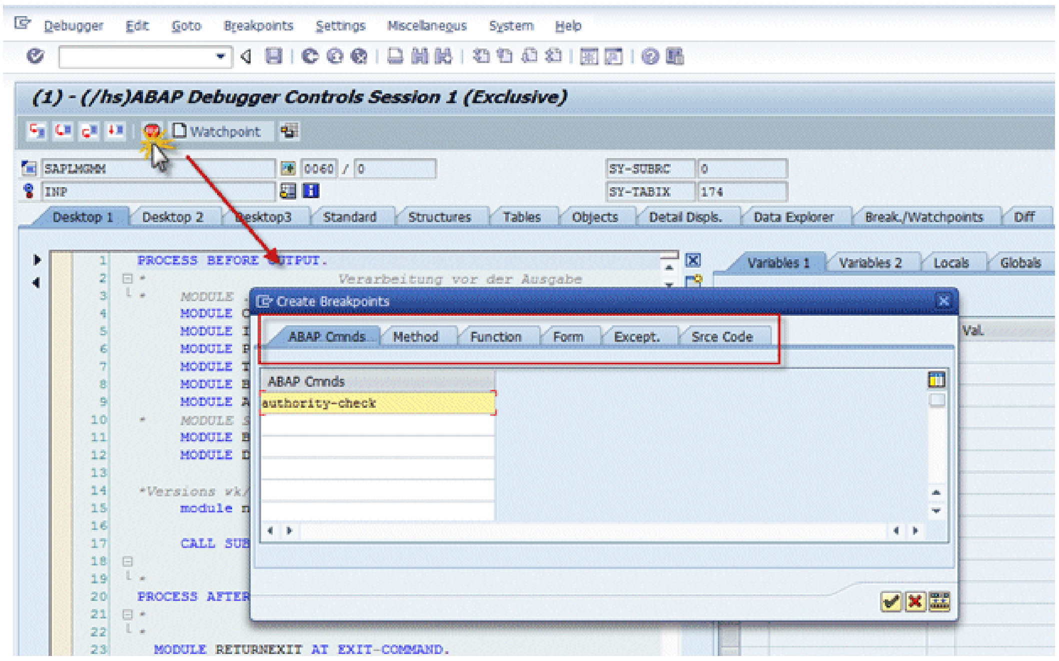
In ABAP there are three different kinds of breakpoints, each of which serve a specific purpose. In releases prior to SAP NetWeaver 2004s, only one breakpoint icon was displayed in the debugger to denote all three types of breakpoints. In SAP NetWeaver 2004s, three different breakpoints icons are displayed in the New ABAP Debugger, as well as in the ABAP Editor, so you know immediately which kind of breakpoint you are dealing with.

You can set **debugger breakpoints** inside the debugger. They exist only as long as the debugger is active. Once the debugger is closed, all debugger breakpoints are gone.

Usually, you set **session breakpoints** in the ABAP Editor. The scope of session breakpoints is the current logon session. This means that your session breakpoints are present in all external sessions (i.e., SAP GUI windows) of the current logon session. If you select the Save button in the ABAP Debugger (in a dialog logon session), all current debugger breakpoints are automatically converted to session breakpoints.

If you set a breakpoint inside the coding of a Web Dynpro or BSP application, then the ABAP Workbench automatically sets a **user breakpoint**. User breakpoints are stored in the SAP database, and they are valid for all logon sessions of the current user on the current application server. If you set a user breakpoint, then all following logon sessions of this user will have those breakpoints set

Setting breakpoints in the New ABAP Debugger is not much different from setting them in the Classic ABAP Debugger. If you click on the breakpoint icon in the control area of the New ABAP Debugger, the Create Breakpoints dialog appears. Here you select whether you want the breakpoint to occur at ABAP commands, methods, functions, or forms, or whenever an exception is caught. You can also set a breakpoint at an *arbitrary* source code position (even in not-yet-loaded programs). This capability is not available in the Classic ABAP Debugger.



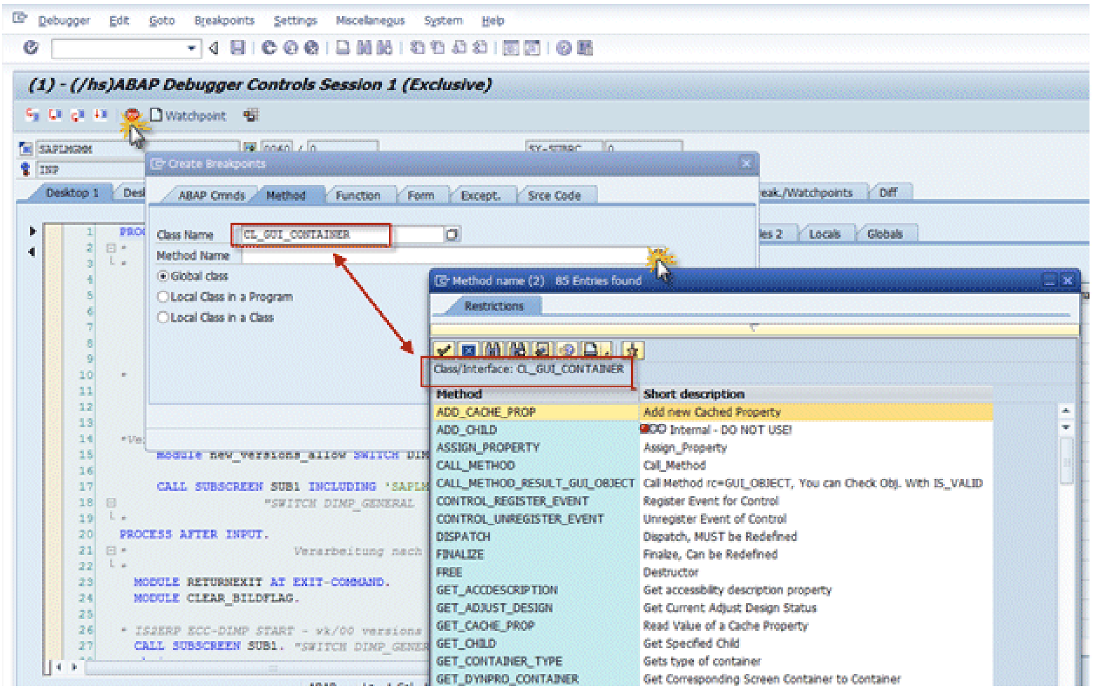
(1) - (/hs)ABAP Debugger Controls Session 1 (Exclusive)

SAPLMGMM	0060 / 0	SY-SUBRC	0
INP		SY-TABIX	174

```
PROCESS BEFORE OUTPUT.
                              Verarbeitung vor der Ausgabe
*   MODULE
    MODULE
    MODULE
    MODULE
    MODULE
    MODULE
    MODULE
*   MODULE
    MODULE
    MODULE

*Versions vk/
    module

    CALL SUB

PROCESS AFTER

    MODULE RETURNEXIT AT EXIT-COMMAND.
```

ABAP Cmnds
authority-check

You can get help when you are setting breakpoints at a method, function, or form by pressing F4. For example, pressing F4 in the Method Name field on the Method tab displays all included methods of a given class. So even if you do not have all the details about the method or function module, you can still set a breakpoint. In the Classic ABAP Debugger, there is no such help available.



(1) - (/hs)ABAP Debugger Controls Session 1 (Exclusive)

Class Name: CL_GUI_CONTAINER
Method Name:
(● Global class) ○ Local Class in a Program ○ Local Class in a Class

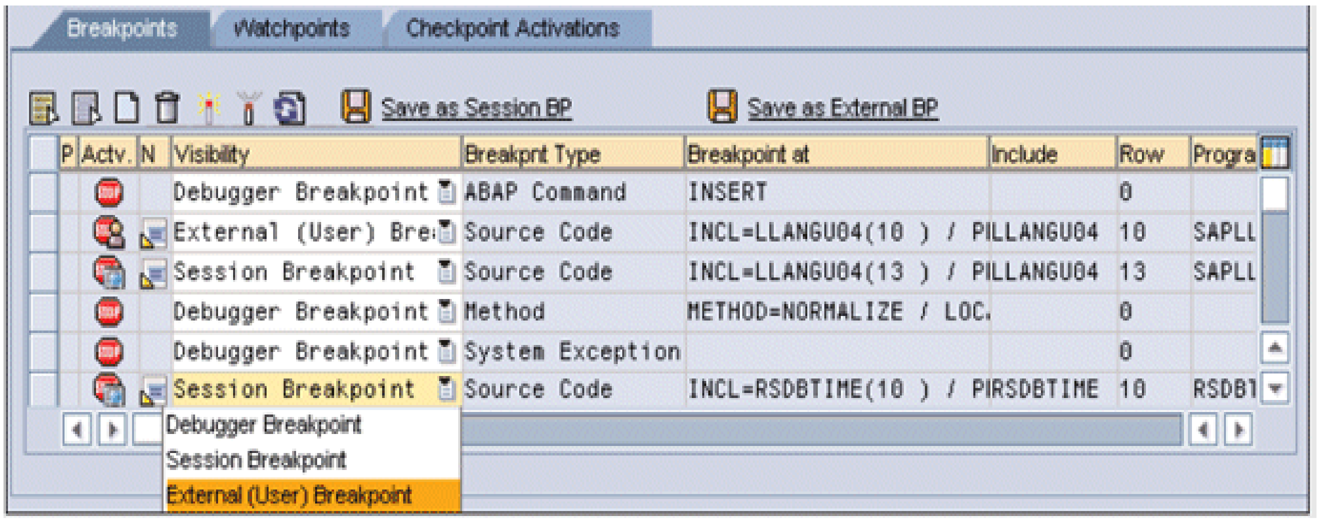
```
PROCESS AFTER INPUT.
    module new_versions_allow_switch DIM
    CALL SUBSCREEN SUB1 INCLUDING 'SAPLM
                    "SWITCH DIMP_GENERAL
PROCESS AFTER INPUT.
                              Verarbeitung nach
    MODULE RETURNEXIT AT EXIT-COMMAND.
    MODULE CLEAR_BILDFLAG.

* IS2ERP ECC-DIMP START - vk/00 versions
    CALL SUBSCREEN SUB1. "SWITCH DIMP_GENER
```

Method name (2)   85 Entries found

Class/Interface: CL_GUI_CONTAINER

Method	Short description
ADD_CACHE_PROP	Add new Cached Property
ADD_CHILD	● OO Internal - DO NOT USE!
ASSIGN_PROPERTY	Assign_Property
CALL_METHOD	Call_Method
CALL_METHOD_RESULT_GUI_OBJECT	Call Method rc=GUI_OBJECT, You can Check Obj. With IS_VALID
CONTROL_REGISTER_EVENT	Register Event for Control
CONTROL_UNREGISTER_EVENT	Unregister Event of Control
DISPATCH	Dispatch, MUST be Redefined
FINALIZE	Finalize, Can be Redefined
FREE	Destructor
GET_ACCDESCRIPTION	Get accessibility description property
GET_ADJUST_DESIGN	Get Current Adjust Design Status
GET_CACHE_PROP	Read Value of a Cache Property
GET_CHILD	Get Specified Child
GET_CONTAINER_TYPE	Gets type of container
GET_DYNPRO_CONTAINER	Get Corresponding Screen Container to Container

Using the new Breakpoints tool, which is on the Break/Watchpoints desktop, you can administer all your currently set breakpoints. You can create, delete, activate, or inactivate breakpoints using this tool. In the New ABAP Debugger, you can treat the breakpoints individually. For example you can promote any single debugger breakpoint to a session breakpoint or any session breakpoint to a user breakpoint..etc



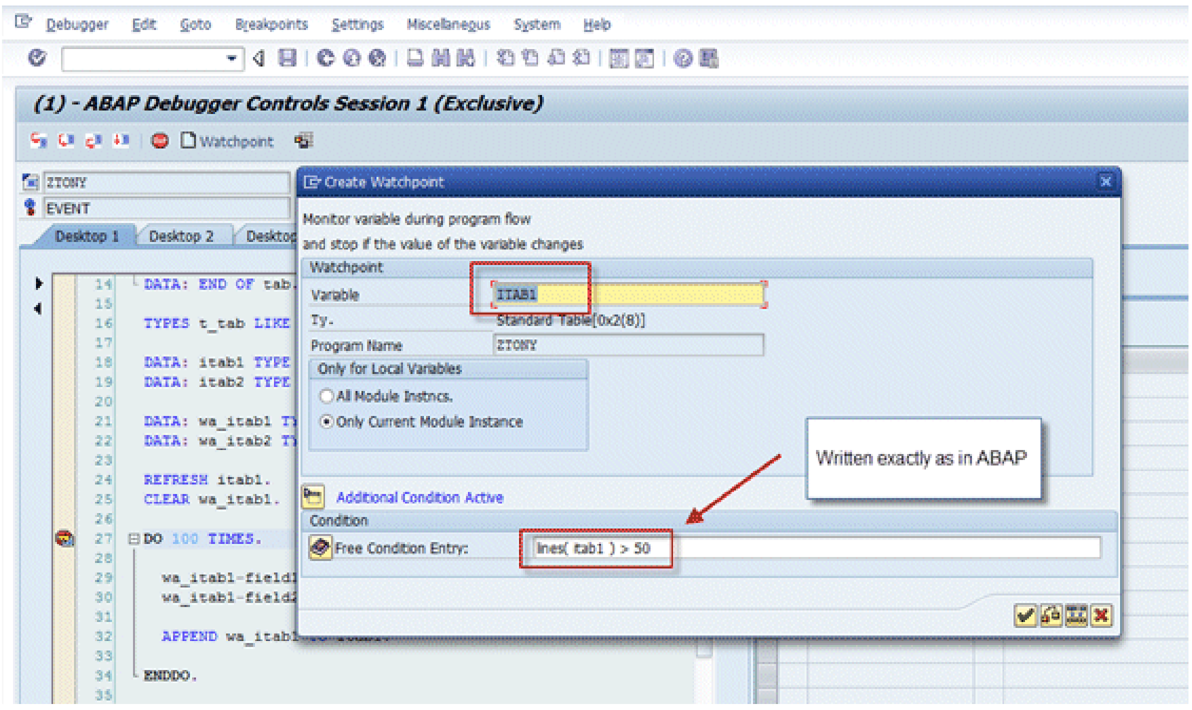
P	Actv.	N	Visibility	Breakprnt Type	Breakpoint at	Include	Row	Progra
	●		Debugger Breakpoint	ABAP Command	INSERT		0	
	●		External (User) Bre	Source Code	INCL=LLANGU04(10) / PILLANGU04		10	SAPLL
	●		Session Breakpoint	Source Code	INCL=LLANGU04(13) / PILLANGU04		13	SAPLL
	●		Debugger Breakpoint	Method	METHOD=NORMALIZE / LOC.		0	
	●		Debugger Breakpoint	System Exception			0	
	●		Session Breakpoint	Source Code	INCL=RSDBTIME(10) / PIRSDBTIME		10	RSDBT

Watch-Points

A watchpoint can be set on any variable with local or global scope to break the execution of the program when the variable's content changes. This makes it easier to debug the program's data flow.

In order to create a watchpoint, click on the Watchpoint button in the control area of the New ABAP Debugger. In the pop-up dialog that appears, you can specify the variable, the scope for a watchpoint on a local variable, and a free condition. The scope for a watchpoint on a local variable means that you can set the watchpoint for only the current instance of the module on the execution stack in which the local variable exists, or for all instances of this module that will be pushed onto the stack. The ability to associate a free condition with the watchpoint is a convenient way to limit the number of "watchpoint reached" events according to your needs. When you specify a condition along with watchpoints, it is a free-style condition for which you can choose two arbitrary operands (no need to specify the watchpoint variable, but you can) and one operator. Write the condition in the same way you would write it in ABAP. You can use two built-in functions as operands in the watchpoint condition….. "lines(*itab*)" and "strlen(*str*)" (the number of lines of the internal table *itab* and the length of the string *str*, respectively).

For example, I will set a watchpoint on the internal table itab1 to break when it is greater than 50 lines in content.



(1) - ABAP Debugger Controls Session 1 (Exclusive)

ZTONY
EVENT

```
DATA: END OF tab.

TYPES t_tab LIKE

DATA: itab1 TYPE
DATA: itab2 TYPE

DATA: wa_itab1 TY
DATA: wa_itab2 TY

REFRESH itabl.
CLEAR wa_itabl.

DO 100 TIMES.

  wa_itabl-field1
  wa_itabl-field2

  APPEND wa_itabl

ENDDO.
```

Create Watchpoint

Monitor variable during program flow
and stop if the value of the variable changes

Watchpoint

Variable	ITAB1
Ty.	Standard Table[0x2(8)]
Program Name	ZTONY

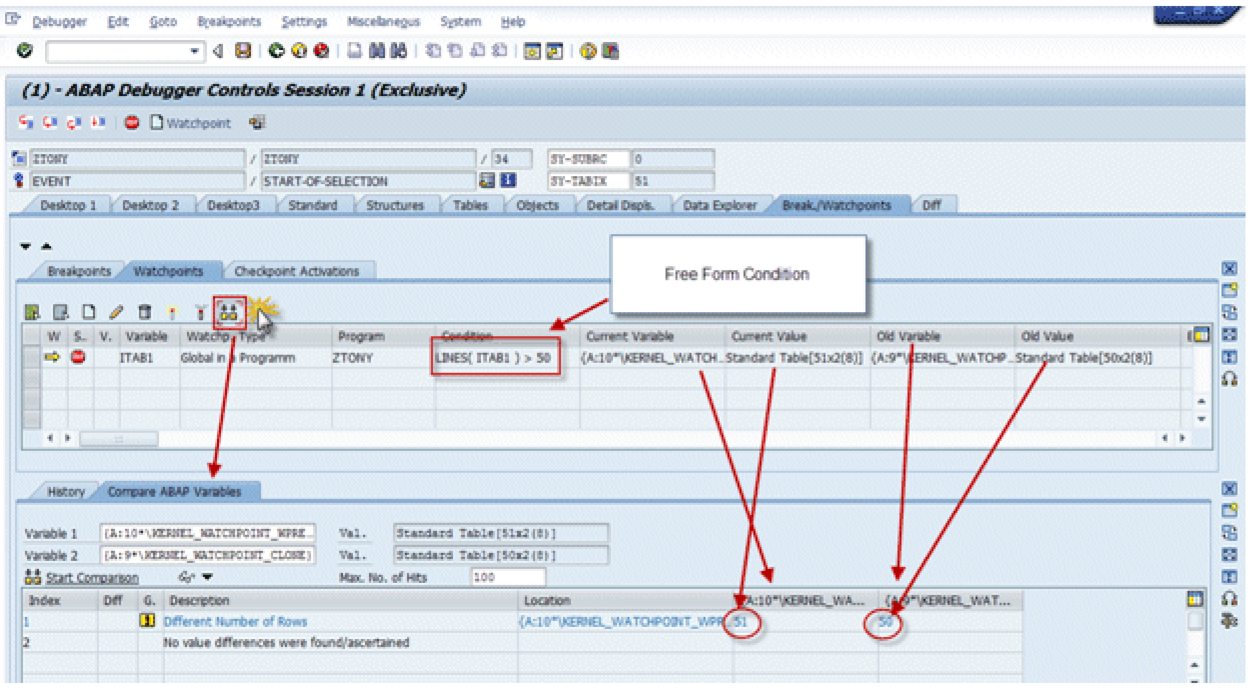
Only for Local Variables
( ) All Module Instncs.
(●) Only Current Module Instance

Additional Condition Active

Condition

Free Condition Entry: lines(itab1) > 50

All currently set watchpoints are listed in the Watchpoints tab of the Breakpoints tool. Here you can create, edit, delete, activate, or inactivate watchpoints. In addition to information such as the variable name, the scope, and the condition, you will find in the watchpoint list a symbol for the "old variable" for all watchpoints. This information enables you to view the value of the variable of the recently hit watchpoint before it was changed. I will execute the above watchpoint for illustration of this point…


Debugger Edit Goto Breakpoints Settings Miscellaneous System Help
Watchpoint
Desktop 1 Desktop 2 Desktop3 Standard Structures Tables Objects Detail Displ. Data Explorer Break./Watchpoints Diff
Breakpoints Watchpoints Checkpoint Activations
Free Form Condition
History Compare ABAP Variables
Start Comparison

(1) - ABAP Debugger Controls Session 1 (Exclusive)

ZTONY	/ ZTONY	/ 34	SY-SUBRC	0
EVENT	/ START-OF-SELECTION		SY-TABIX	51

W	S..	V.	Variable	Watchp. Type	Program	Condition	Current Variable	Current Value	Old Variable	Old Value
			ITAB1	Global in Programm	ZTONY	LINES(ITAB1) > 50	{A:10*\KERNEL_WATCH…	Standard Table[51x2(8)]	{A:9*\KERNEL_WATCHP…	Standard Table[50x2(8)]

Variable 1	{A:10*\KERNEL_WATCHPOINT_WPRE…	Val.	Standard Table[51x2(8)]
Variable 2	{A:9*\KERNEL_WATCHPOINT_CLONE}	Val.	Standard Table[50x2(8)]
		Max. No. of Hits	100

Index	Diff	G.	Description	Location	{A:10*\KERNEL_WA…	{A:9*\KERNEL_WAT…
1			Different Number of Rows	{A:10*\KERNEL_WATCHPOINT_WPR…	51	50
2			No value differences were found/ascertained			

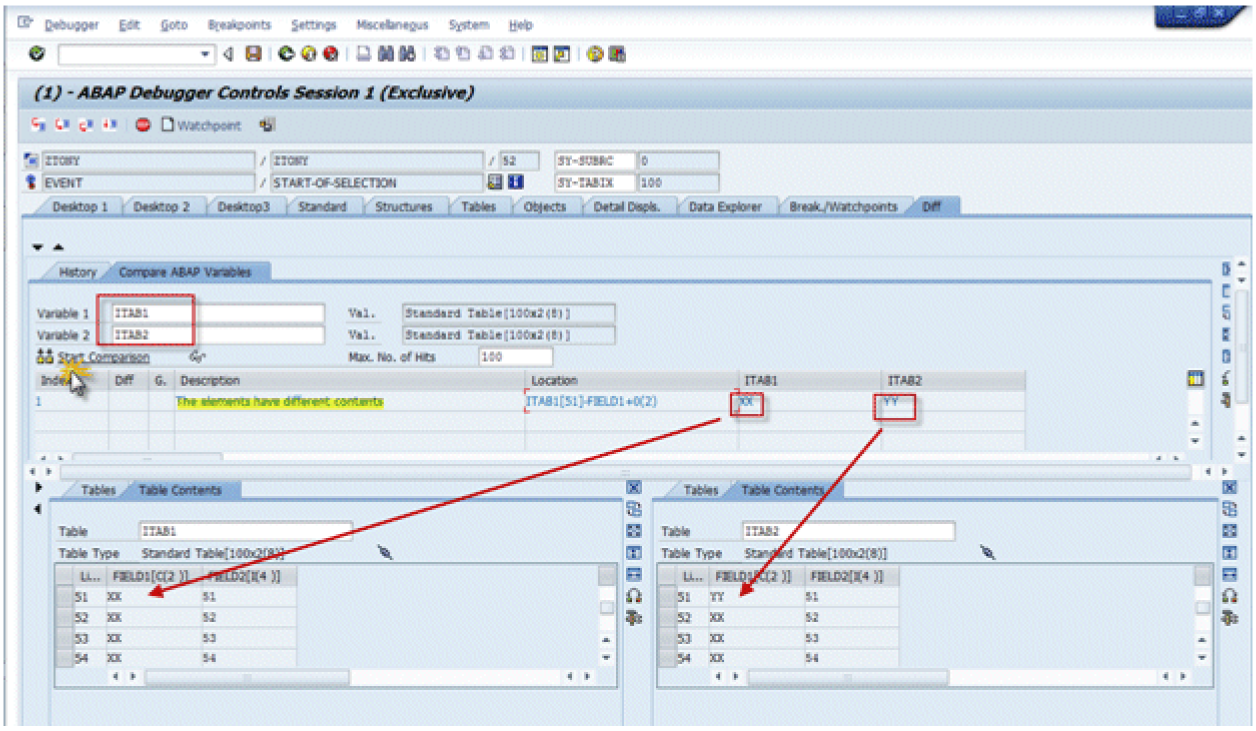
The final tool I want to explore is the new **DIFF tool.**

Do you remember this situation…

You are hot on the trail and need to find the differences between two tables. Using the Tables desktop, we can easily create two table views in order to compare the two tables. However, for large internal tables, finding out the differences within an acceptable period of time is not an easy task.

The New ABAP Debugger provides a special tool for this task — the DIFF Tool, which finds the differences between two arbitrary ABAP data objects. You can compare internal tables, structures, strings, or even objects using the DIFF Tool, which provides the differences concerning the types (e.g., one table is a sorted table and the other a hashed table), and of course, the values.

I will start the DIFF Tool for the internal tables' itab1 and itab2; simply click on the Start Comparison button, and viola, a complete display showing a list of all differences between the two tables.



(1) - ABAP Debugger Controls Session 1 (Exclusive)

ZTOHY	/	ZTOHY	/ 52	SY-SUBRC	0
EVENT	/	START-OF-SELECTION		SY-TABIX	100

Variable 1	ITAB1	Val.	Standard Table[100x2(8)]
Variable 2	ITAB2	Val.	Standard Table[100x2(8)]
		Max. No. of Hits	100

Index	Diff	G.	Description	Location	ITAB1	ITAB2
1			The elements have different contents	ITAB1[51]-FIELD1+0(2)	XX	YY

Table ITAB1
Table Type Standard Table[100x2(8)]

Li...	FIELD1[C(2)]	FIELD2[I(4)]
51	XX	51
52	XX	52
53	XX	53
54	XX	54

Table ITAB2
Table Type Standard Table[100x2(8)]

Li...	FIELD1[C(2)]	FIELD2[I(4)]
51	YY	51
52	XX	52
53	XX	53
54	XX	54

If you enjoyed this ebook and want to learn more, click the button below. Learn about the WebDynpro Debugging tool, how to run ABAP traces inside the debugger, and all about Software Layering (SLAD)

Click the button now!

Learn more