ABAP Debugger (/h) Overview

ABAP Debugger (/h)

Why is it needed?

Almost all SAP applications use ABAP. Therefore if you want to analyze them deeply then you will need the ABAP-debugger.

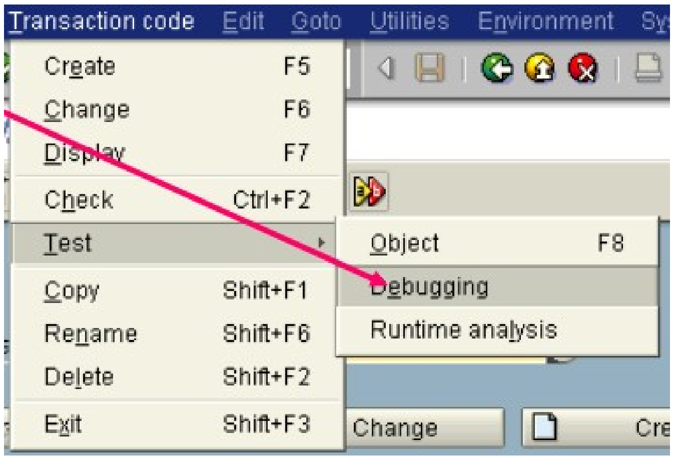
I. Debugger Fundamentals

Ways to start the debugger at the beginning of a program

So before we can even use it, we have to know how to start up the debugger.

- **Report** can be started in debugging mode in the **ABAP Editor (SE38)** by clicking the **debugging button** **.**
- A **function module** can be started in debugging mode in the **Function Builder (SE37)** also be clicking the **debugging button**

  when attempting to test the program.
-
  - This is the only option for debugging **CONVERSION EXITS** or **FIELD EXITS** as the debugger skips over them during a normal debugging session. *****
  - If a program is highly complicated and you have narrowed down the error to a FM you can do isolated tests this way in SE37
- A **transaction** can also be debugged using **Maintain Transaction (SE93)** by clicking in the menu: **Transaction Code->Test->Debugging**

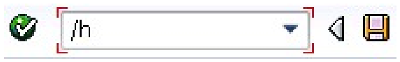
-
  -

- The transaction will start in the the first **Process Before Output (PBO)** module of the first screen of the application
- Same options can be reached from **Object Navigator (SE80)** by clicking the "Other Object" button and filling in the transaction code. This will pull up data like SE93.

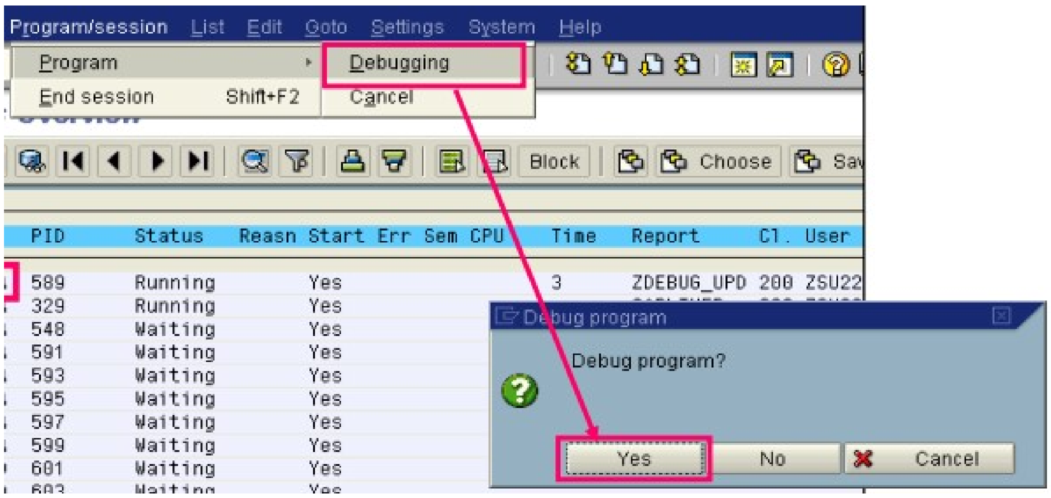
Ways to start the debugger in an already running application

Starting the debugger in an already running application can allow you to get straight to the problem (generally don't need to debug the selection screen of a report)

- For a **transaction** you can start the debugger by using the tcode **/h** for debugging or **/hs** for system debugging



- For basis programs /hs will likely be needed
- For a **Batch Job** or **long running program** you can start the debugger from the **Process Overview (SM50)** transaction.



```
PID  Status   Reasn Start Err Sem CPU  Time  Report     Cl. User
589  Running        Yes                3     ZDEBUG_UPD 200 ZSU22
329  Running        Yes
548  Waiting        Yes
591  Waiting        Yes
593  Waiting        Yes
595  Waiting        Yes
597  Waiting        Yes
599  Waiting        Yes
601  Waiting        Yes
```

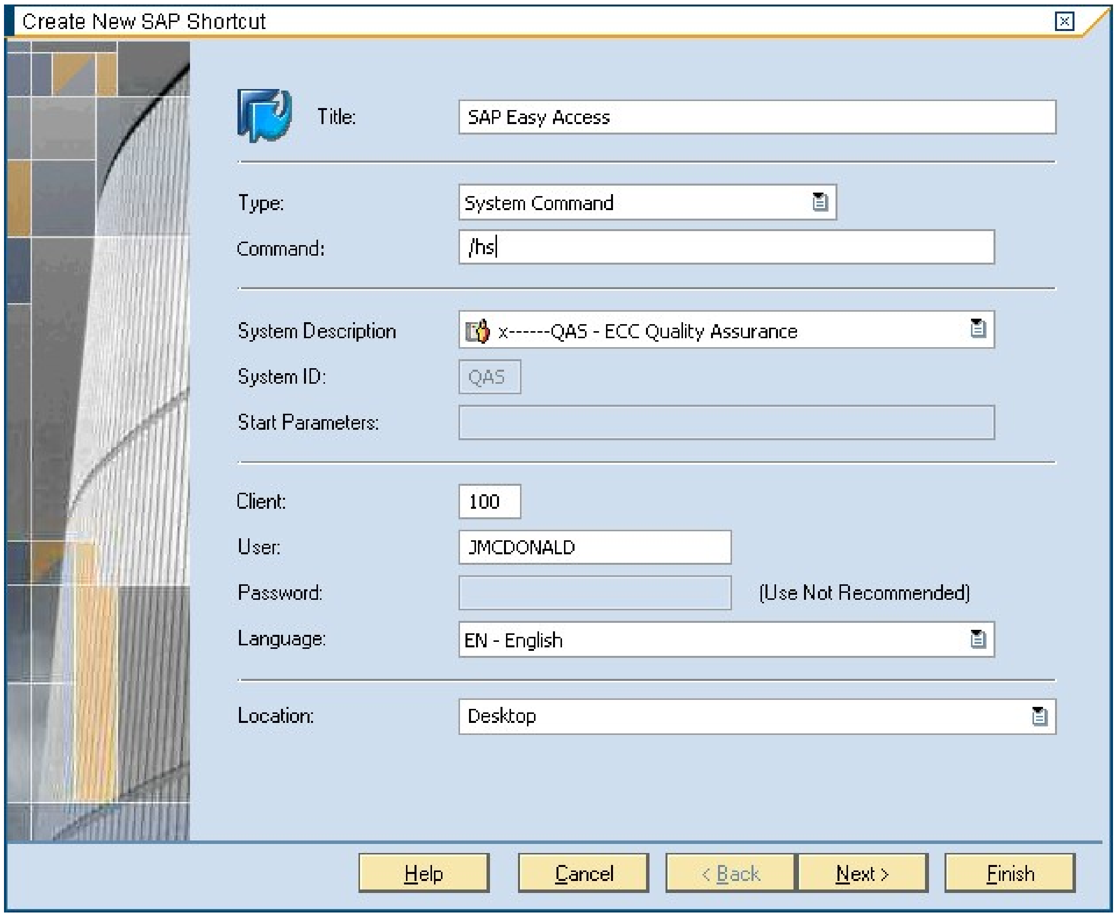
Debug program?

- When debugging a batch process breakpoints will not work. Only an entry in the system log (Break Point Reached) will be processed. To stop a batch processes where you want it to be you must:
  - Create an endless loop in the application prior to the code you want to look at like so:

```
while i = 0.
   i = 0.
endwhile.
```

  - Then start debugging in from SM50.
  - Change i to equal anything other than 0 and you will be able to use the full functionality of the ABAP debugger for the batch process (including breakpoints).
- Sometimes an application may create a **message dialog (popup window)** that does not give you access to the OK-Code to put in /h or /hs but you want to know what is calling the popup or what happens after the popup. For this case we'll have to create a **debugger shortcut.**
  - **Easiest way to do this is to create a .txt file with the following in it**
  - **[Function]**
    Command=/H
    Type=SystemCommand
  - **Then just drag and drop that on the pop up (the shortcut thing below works too. but it's more trouble to create)**

- ◦ From most any screen click the Generate Shortcut button .
  ◦ This will bring up the Create New SAP Shortcut dialog
  ◦



Create New SAP Shortcut

Title:	SAP Easy Access
Type:	System Command
Command:	/hs
System Description	x------QAS - ECC Quality Assurance
System ID:	QAS
Start Parameters:	
Client:	100
User:	JMCDONALD
Password:	(Use Not Recommended)
Language:	EN - English
Location:	Desktop

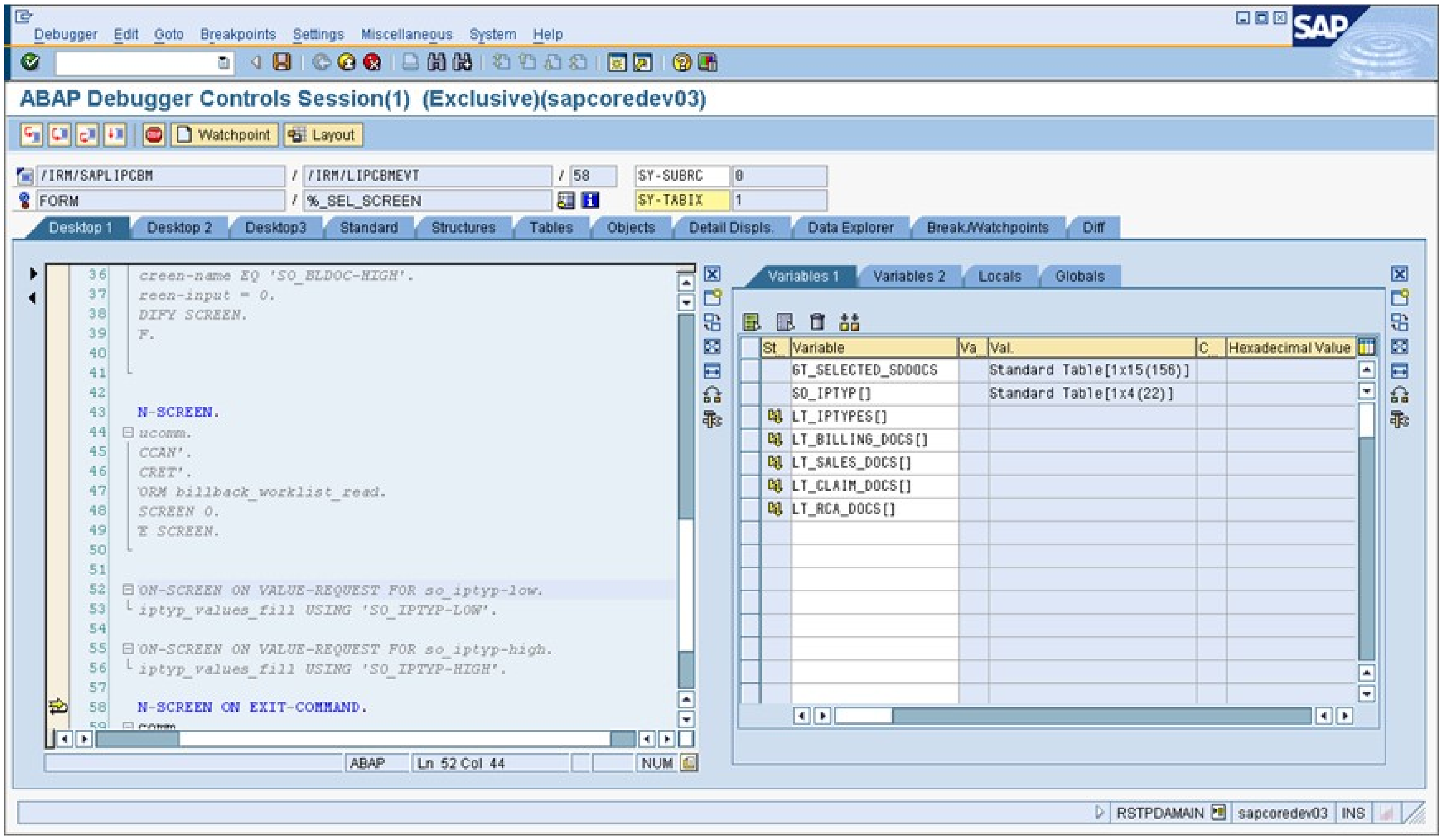
  ◦
  ◦ Simply change the command field to have the /h or /hs t-code and set Type to System Command.
  ◦ The System Description field should not limit you to a specific system. If the shortcut were double clicked on the desktop it would attempt to open in this system but this is not how we will use this shortcut.
  ◦ Save the shortcut to your desktop.
  ◦ Now when you need to debug a message dialog you can drag the shortcut onto the dialog box and it will turn on debugging from that point on.
  ◦ For CITRIX connections make sure you have permission to do this. Additionally, a Windows Explorer window may need to be opened that is on the same connection as your SAP GUI session.
- And last but not least, **Breakpoints.**
- ◦ **Static Breakpoints** are created in code.
  ◦
    ◦ **BREAK-POINT** will cause the debugger to activate during the processing of an application. This breakpoint will stop for all users of the transaction though.
    ◦ **BREAK username** will break for only a specific user.
    ◦
      ◦ For example if user JMCDONALD wants to jump into debugger at a certain part of the code he could add BREAK JMCDONALD and the debugger would trip for only him at that breakpoint
  ◦ **Dynamic Breakpoints** can be created without modifying the code.
  ◦
    ◦ Two kinds for use prior to opening the debugger
    ◦

        - **Session Breakpoints** stays active for a specific user for the entirety of the users session (until they logout).
        - **User Breakpoints** has a 2 hour interval it is active. This is active for all users logged onto the current server of the current system it was created in.
        - **Debugger Breakpoints can be set during a debugger session. These will expire after the debugger is closed and only apply to the user utilizing the debugger.**
            - If the save button is clicked in the debugger these will be converted **to Session Breakpoints.**
    - **Watchpoints**
        - Watch points can be created to monitor variables.
            - The basic creation stops the program whenever a change is detected on a watchpoint
            - Conditions can also be added to a watch point to cause them to only stop execution when a field equals a specific value or other condition.
-

Using the debugger (at last!)

Ah, finally...

Here's a quick shot of what the **New Debugger** can look like:



ABAP Debugger Controls Session(1) (Exclusive)(sapcoredev03)

/IRM/SAPLIPCBM	/ /IRM/LIPCBMEVT	/ 58	SY-SUBRC	0
FORM	/ %_SEL_SCREEN		SY-TABIX	1

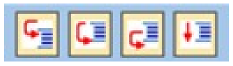
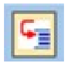
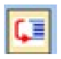
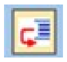
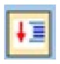
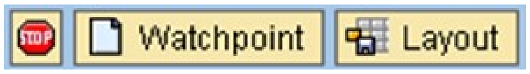
```
  creen-name EQ 'SO_BLDOC-HIGH'.
  reen-input = 0.
  DIFY SCREEN.
  F.



  N-SCREEN.
  ucomm.
  CCAN'.
  CRET'.
  ORM billback_worklist_read.
  SCREEN 0.
  E SCREEN.


  ON-SCREEN ON VALUE-REQUEST FOR so_iptyp-low.
  iptyp_values_fill USING 'SO_IPTYP-LOW'.

  ON-SCREEN ON VALUE-REQUEST FOR so_iptyp-high.
  iptyp_values_fill USING 'SO_IPTYP-HIGH'.

  N-SCREEN ON EXIT-COMMAND.
```

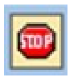
ABAP | Ln 52 Col 44 | NUM

St	Variable	Va	Val.	C	Hexadecimal Value
	GT_SELECTED_SDDOCS		Standard Table[1x15(156)]		
	SO_IPTYP[]		Standard Table[1x4(22)]		
	LT_IPTYPES[]				
	LT_BILLING_DOCS[]				
	LT_SALES_DOCS[]				
	LT_CLAIM_DOCS[]				
	LT_RCA_DOCS[]				

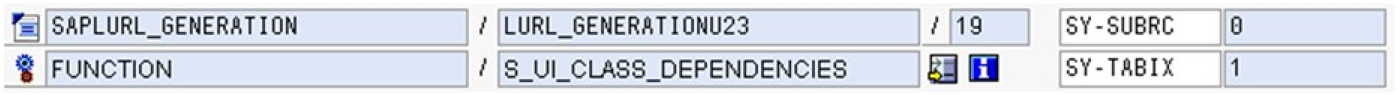
RSTPDAMAIN | sapcoredev03 | INS

Buttons and basic data:

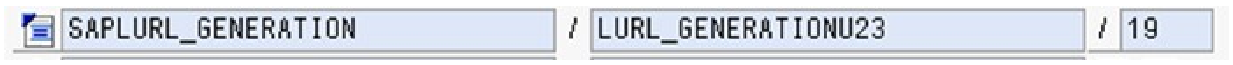
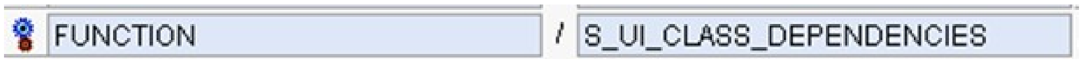
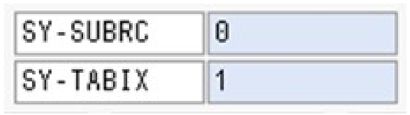
-  - These buttons are used for stepping through the code.
-
  -  - Single Step - Execute single line of code
  -  - Execute - execute any forms/functions/methods without stepping into their execution
  -  - Return - Steps up to calling program from form/function/method
  -  - Continue - Continue until end of processing
  -
    - Break points will cause all of these to stop
- 

-
  - 
    - Create a new breakpoint on an ABAP command
  -

    - Create a new Watchpoint (more on this below)
  -

    - allows you to save the current layout or load previous loads of the debugger.
- 

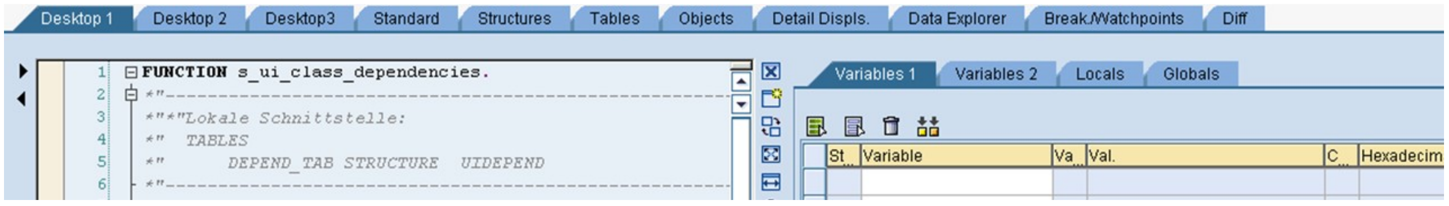
  SAPLURL_GENERATION / LURL_GENERATIONU23 / 19 SY-SUBRC 0

  FUNCTION / S_UI_CLASS_DEPENDENCIES SY-TABIX 1
  - 
    SAPLURL_GENERATION / LURL_GENERATIONU23 / 19
    - Current program name, current include, and current line of code
  - 
    FUNCTION / S_UI_CLASS_DEPENDENCIES
    - Event type and event name
  - 
    SY-SUBRC 0
    SY-TABIX 1
    - System fields - These can be changes to other fields but these two are pretty useful
  - - Navigate to current position in code and display program attributes buttons

Work areas and display

Adding and removing Work areas and their uses

In the new debugger there are several tabs and work areas that can be used as seen here:



```
FUNCTION s_ui_class_dependencies.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  TABLES
*"      DEPEND_TAB STRUCTURE  UIDEPEND
*"----------------------------------------------------------------------
```

When you first open up the system, Desktop 1 should contain the **Source Code(edit control)** and **Variable Fast Display** work areas.

Each work area should have a few options for editing:

- Changes the size of the work area

- Close work area

- Add a new work area

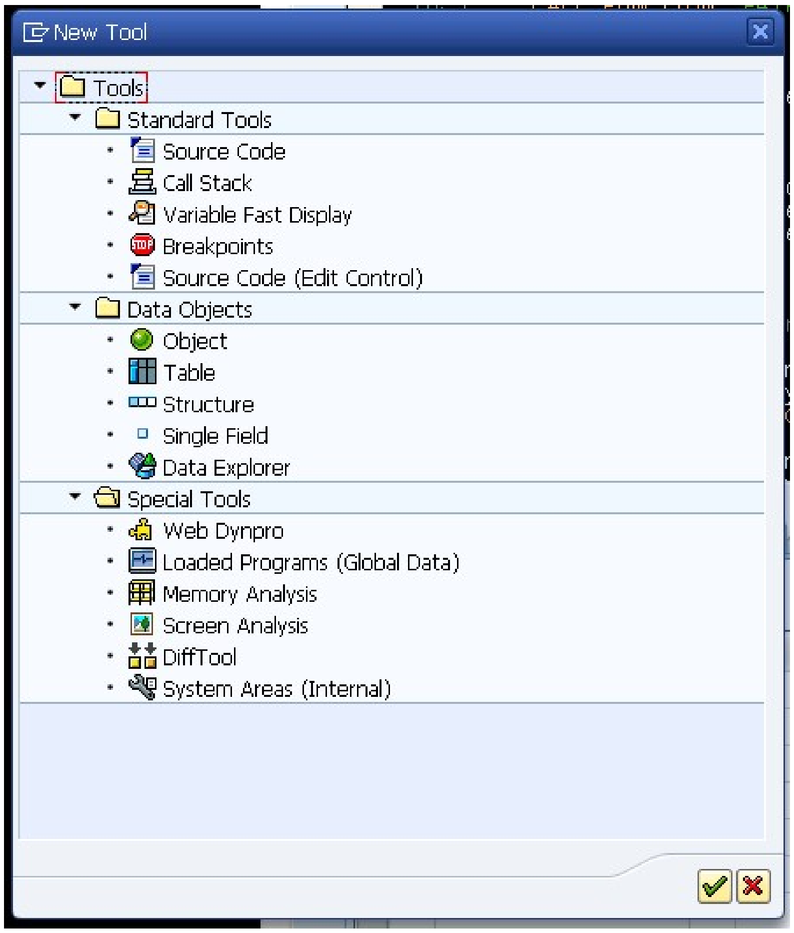
- Replace work area

- Full screen work area,maximize vertically or horizontally

- Swap work areas

- Work area specific tools

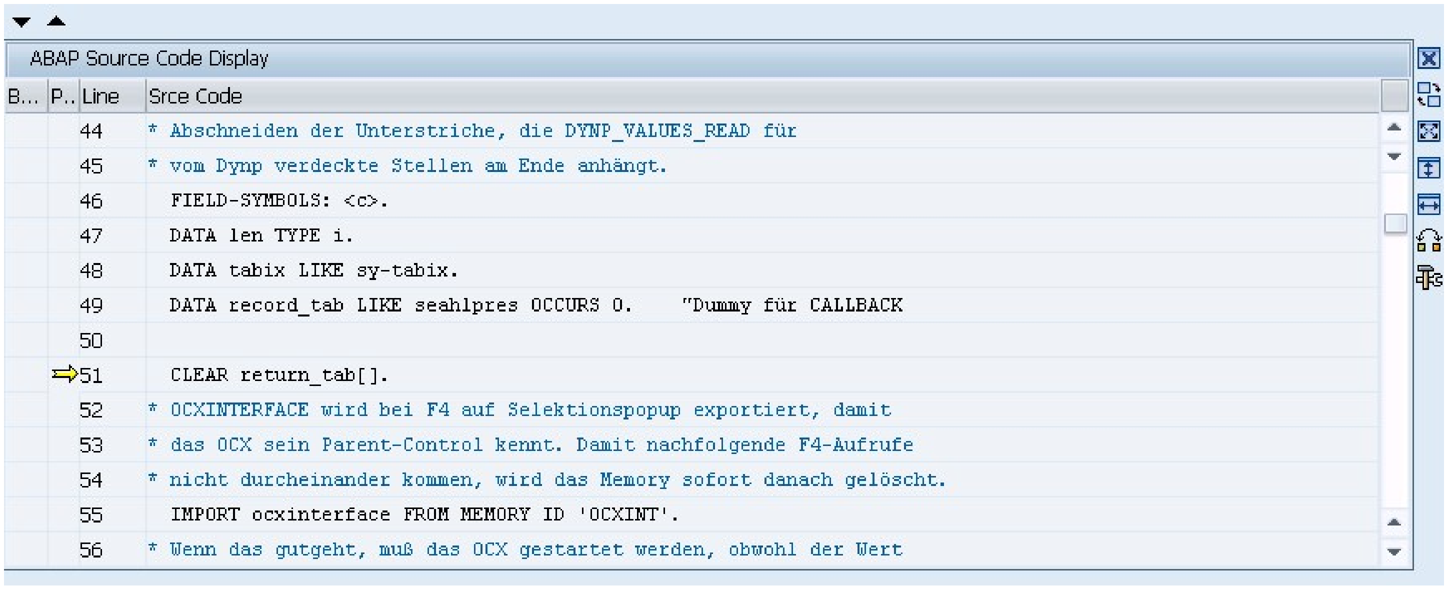
So while the default may just give you the source code and variable display, you have the option to change this however you want. For instance, if you are debugging and following the details of a specific structure or table you can open the related work areas to display these full time instead of having to access them through the variable fast display work area.



Let's explore the different work areas that are available.

Tools

Source Code (old)



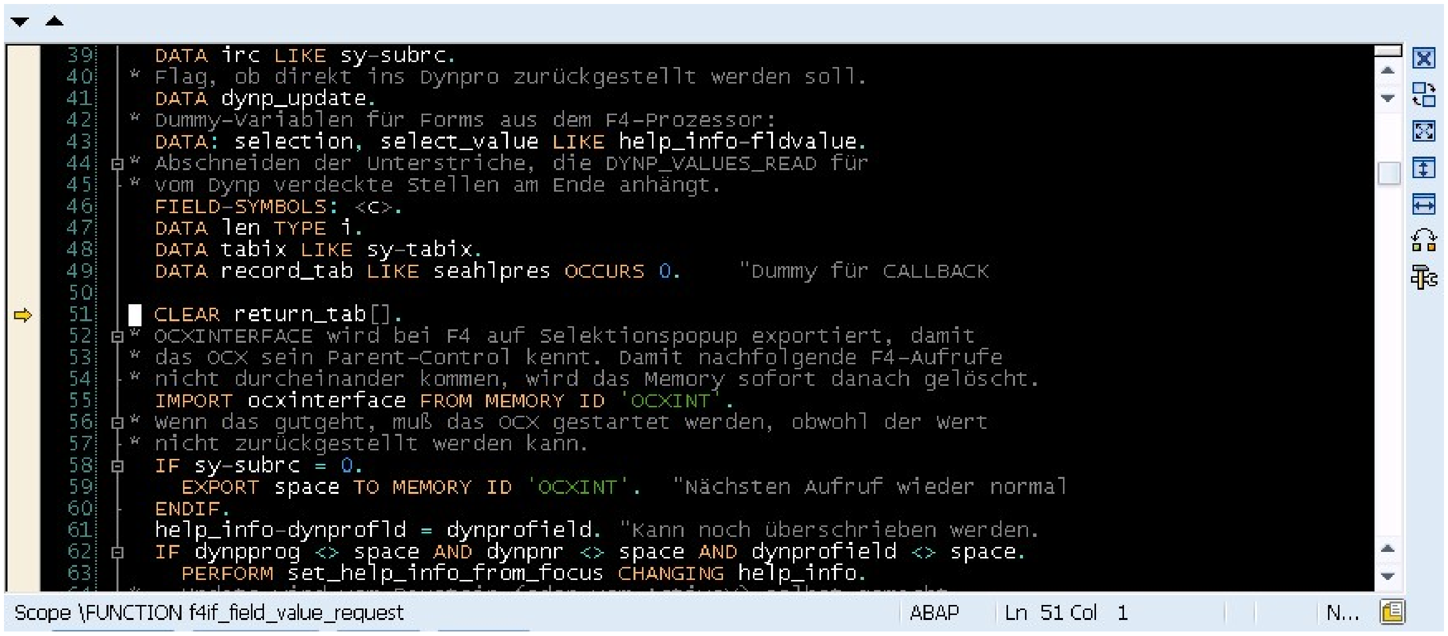
ABAP Source Code Display

B... P.. Line Srce Code

```
* Abschneiden der Unterstriche, die DYNP_VALUES_READ für
* vom Dynp verdeckte Stellen am Ende anhängt.
  FIELD-SYMBOLS: <c>.
  DATA len TYPE i.
  DATA tabix LIKE sy-tabix.
  DATA record_tab LIKE seahlpres OCCURS 0.    "Dummy für CALLBACK

  CLEAR return_tab[].
* OCXINTERFACE wird bei F4 auf Selektionspopup exportiert, damit
* das OCX sein Parent-Control kennt. Damit nachfolgende F4-Aufrufe
* nicht durcheinander kommen, wird das Memory sofort danach gelöscht.
  IMPORT ocxinterface FROM MEMORY ID 'OCXINT'.
* Wenn das gutgeht, muß das OCX gestartet werden, obwohl der Wert
```

This is the source code viewer typical to the Classical Debugger. Not quite as pretty as the new style.

Source Code (edit control)

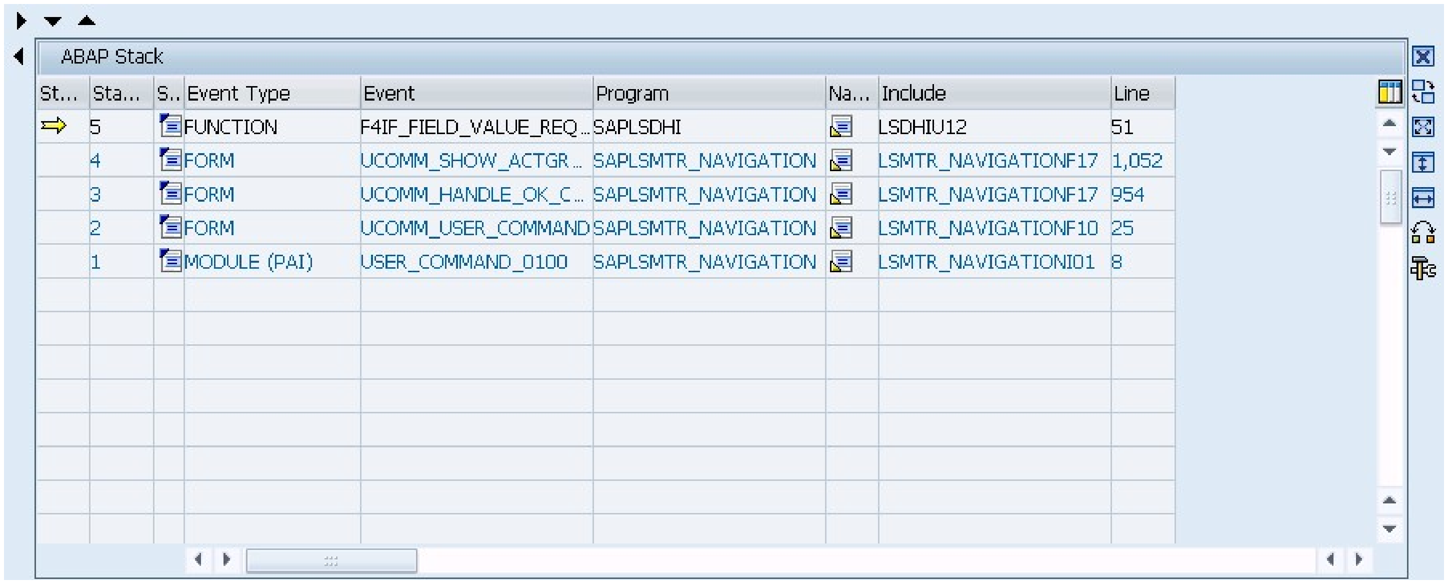


```
  DATA irc LIKE sy-subrc.
* Flag, ob direkt ins Dynpro zurückgestellt werden soll.
  DATA dynp_update.
* Dummy-Variablen für Forms aus dem F4-Prozessor:
  DATA: selection, select_value LIKE help_info-fldvalue.
* Abschneiden der Unterstriche, die DYNP_VALUES_READ für
* vom Dynp verdeckte Stellen am Ende anhängt.
  FIELD-SYMBOLS: <c>.
  DATA len TYPE i.
  DATA tabix LIKE sy-tabix.
  DATA record_tab LIKE seahlpres OCCURS 0.    "Dummy für CALLBACK

  CLEAR return_tab[].
* OCXINTERFACE wird bei F4 auf Selektionspopup exportiert, damit
* das OCX sein Parent-Control kennt. Damit nachfolgende F4-Aufrufe
* nicht durcheinander kommen, wird das Memory sofort danach gelöscht.
  IMPORT ocxinterface FROM MEMORY ID 'OCXINT'.
* Wenn das gutgeht, muß das OCX gestartet werden, obwohl der Wert
* nicht zurückgestellt werden kann.
  IF sy-subrc = 0.
    EXPORT space TO MEMORY ID 'OCXINT'.  "Nächsten Aufruf wieder normal
  ENDIF.
  help_info-dynprofld = dynprofield. "Kann noch überschrieben werden.
  IF dynpprog <> space AND dynpnr <> space AND dynprofield <> space.
    PERFORM set_help_info_from_focus CHANGING help_info.
```

Scope \FUNCTION f4if_field_value_request   ABAP   Ln 51 Col 1   N...

Looks much better (Note this is my personal display settings and I'm color blind so if it looks wonky then don't mind it to bad ok?) and uses the source code editor as the display. This means you get syntax highlighting, easier to scroll around, and you can hover over a variable and get a quick display of it's values. Plus debugger break points can be set easily by clicking the margin.

Call Stack



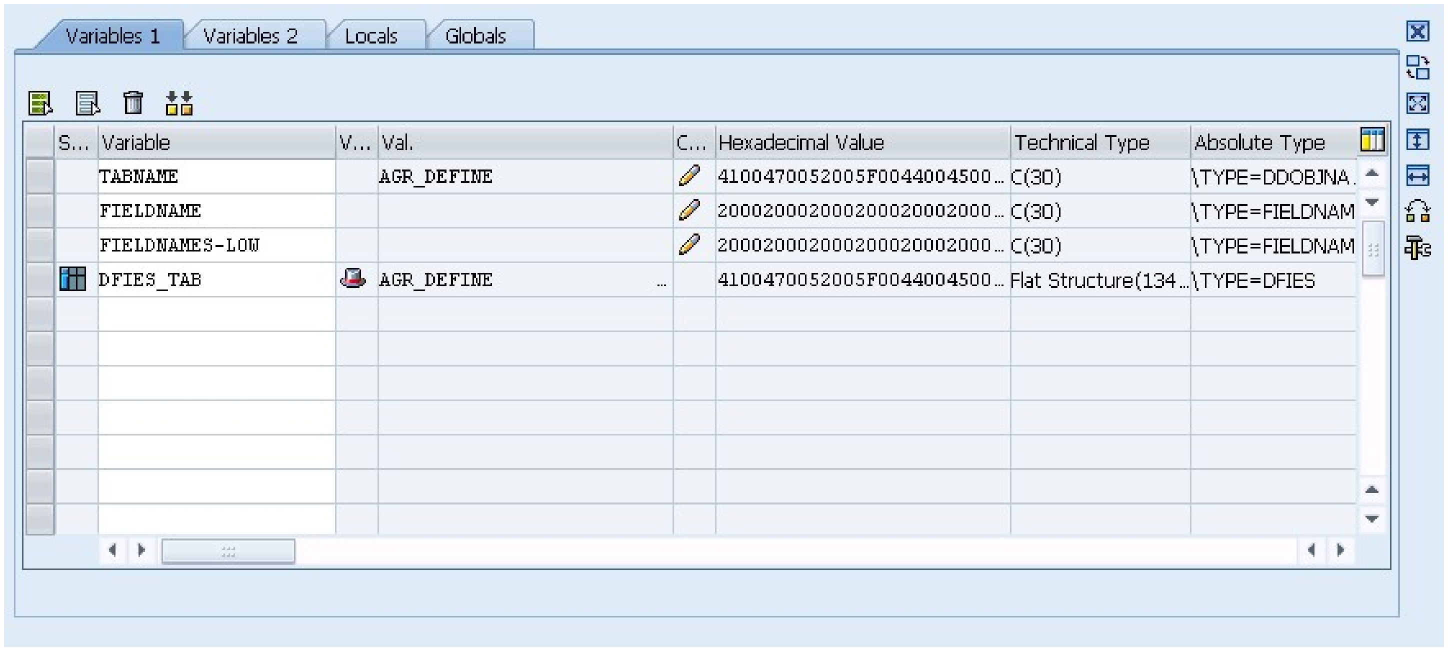
ABAP Stack

St...	Sta...	S..	Event Type	Event	Program	Na...	Include	Line
⇨	5		FUNCTION	F4IF_FIELD_VALUE_REQ...	SAPLSDHI		LSDHIU12	51
	4		FORM	UCOMM_SHOW_ACTGR...	SAPLSMTR_NAVIGATION		LSMTR_NAVIGATIONF17	1,052
	3		FORM	UCOMM_HANDLE_OK_C...	SAPLSMTR_NAVIGATION		LSMTR_NAVIGATIONF17	954
	2		FORM	UCOMM_USER_COMMAND	SAPLSMTR_NAVIGATION		LSMTR_NAVIGATIONF10	25
	1		MODULE (PAI)	USER_COMMAND_0100	SAPLSMTR_NAVIGATION		LSMTR_NAVIGATIONI01	8

This view allows you to see the current call hierarchy (which program was called by which program). The buttons open up an ABAP editor to display the line of code that called the item in the stack above it.

Keep in mind this is the simple call heirarchy. If a program jumped to another routine and returned already you will not find it in the call stack. For this you can use the **ABAP Trace (SE30)** discussed in another note. If system debugging is enabled you can jump to other levels of the call stack

Variable Fast Display



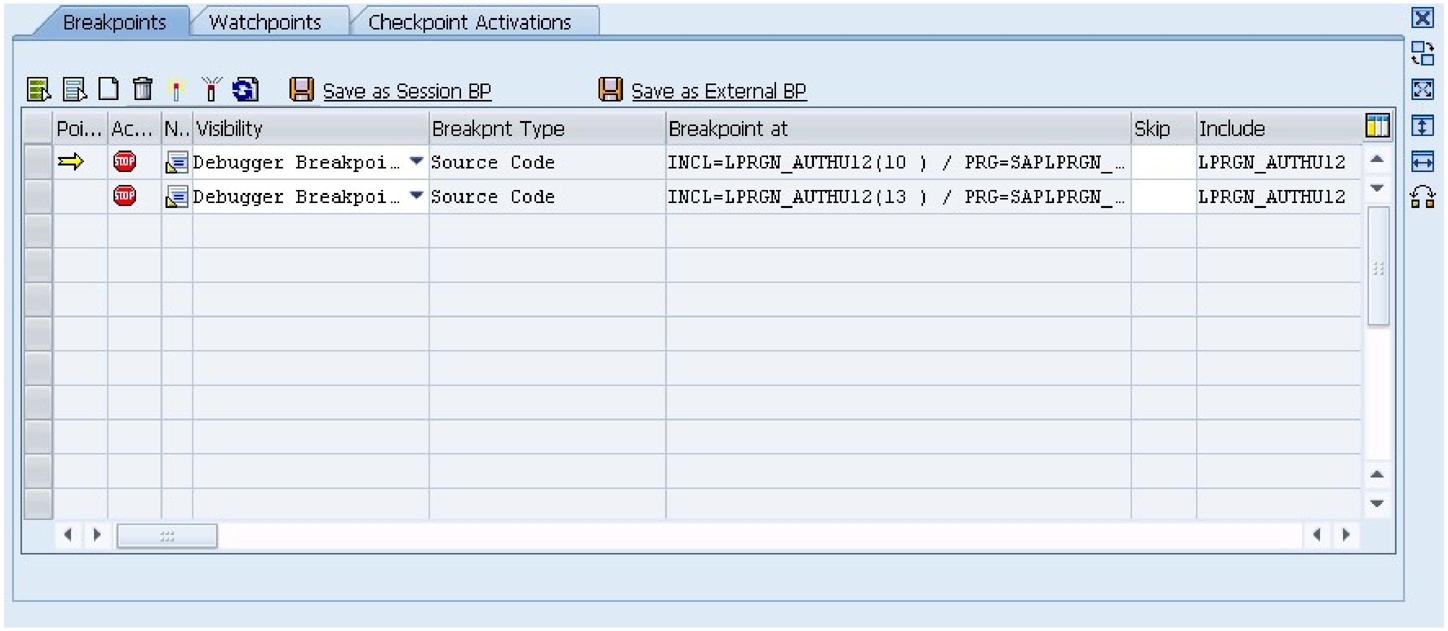
S...	Variable	V...	Val.	C...	Hexadecimal Value	Technical Type	Absolute Type
	TABNAME		AGR_DEFINE		4100470052005F0044004500...	C(30)	\TYPE=DDOBJNA...
	FIELDNAME				2000200020002000200020002000...	C(30)	\TYPE=FIELDNAM
	FIELDNAMES-LOW				2000200020002000200020002000...	C(30)	\TYPE=FIELDNAM
	DFIES_TAB		AGR_DEFINE ...		4100470052005F0044004500...	Flat Structure(134...	\TYPE=DFIES

The Variables 1 & 2 allows you to display variables that are currently being used by the program. You can either type in the variable names yourself or double click them in the source code editor. This view shows you the value of the variable and also allows you to click on internal tables and structures to view their contents with the proper work areas.

The locals and globals tab display a full list of local and global variables that the program has access to.

This view also allows you to edit the values of a variable. For instance, by clicking the in the line for TABNAME, you open up the value for editing. The same can be done with the values in a structure or internal table in their respective work areas.

Breakpoints



Poi...	Ac...	N..	Visibility	Breakpnt Type	Breakpoint at	Skip	Include
⇨	🛑		Debugger Breakpoi...	Source Code	INCL=LPRGN_AUTHU12(10) / PRG=SAPLPRGN_...		LPRGN_AUTHU12
	🛑		Debugger Breakpoi...	Source Code	INCL=LPRGN_AUTHU12(13) / PRG=SAPLPRGN_...		LPRGN_AUTHU12

The breakpoints tab displays all breakpoints that related to the code the user is debugging and can affect the current user. It allows you to modify and delete the active breakpoints specifically. You can also assign a counter to skip the breakpoint after so many hits.

The watchpoints tab displays all currently created watchpoints (more on creating watch points later).

Data Objects

The data objects section of the work area tools provides specific work areas for specific data objects. This is generally self explanatory here.

Special Tools

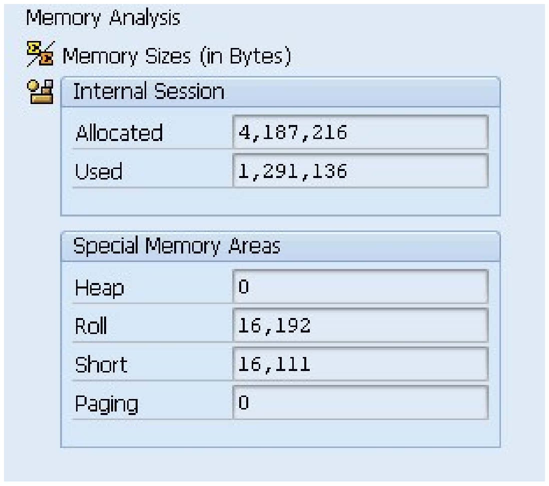
Web Dynpro

Web Dynpro Debugger

No Web Dynpro Runtime Active

Displays 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. Don't know what any of that means. Never used it before!

Memory Analysis



Memory Analysis

Memory Sizes (in Bytes)

Internal Session

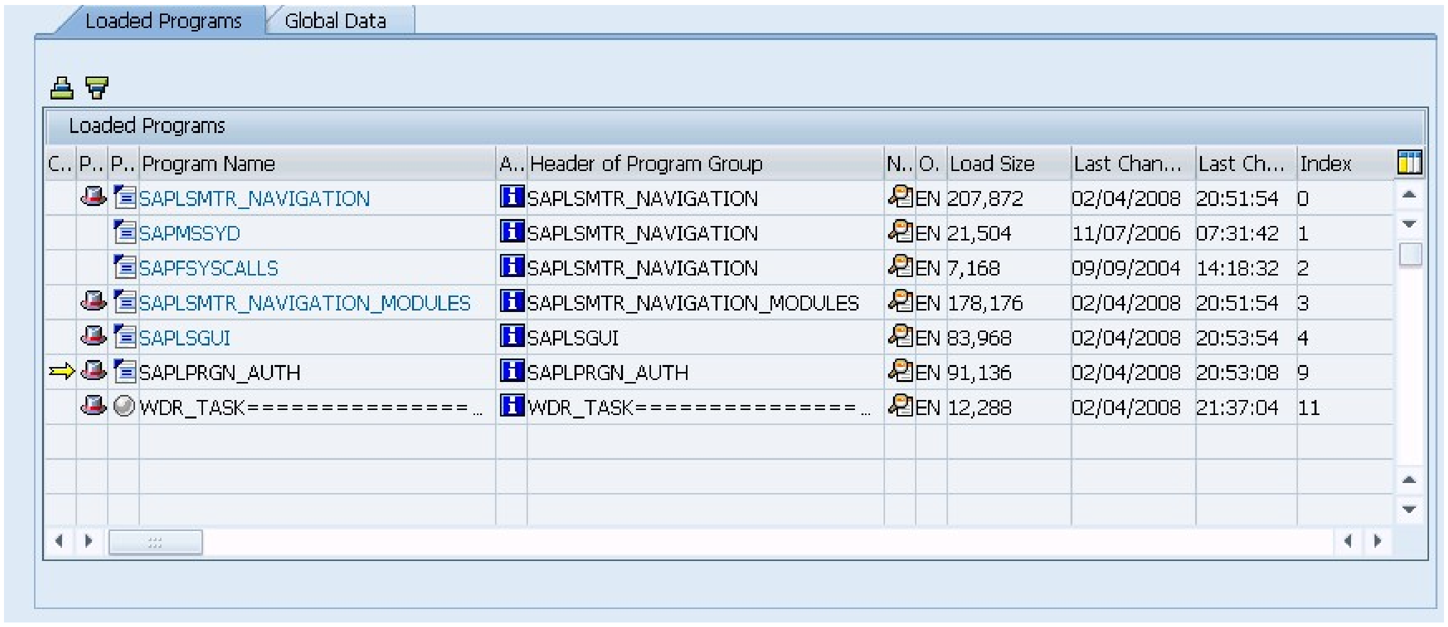
Allocated	4,187,216
Used	1,291,136

Special Memory Areas

Heap	0
Roll	16,192
Short	16,111
Paging	0

Displays the Memory Inspector tool (S_MEMORY_INSPECTOR)

Loaded Programs

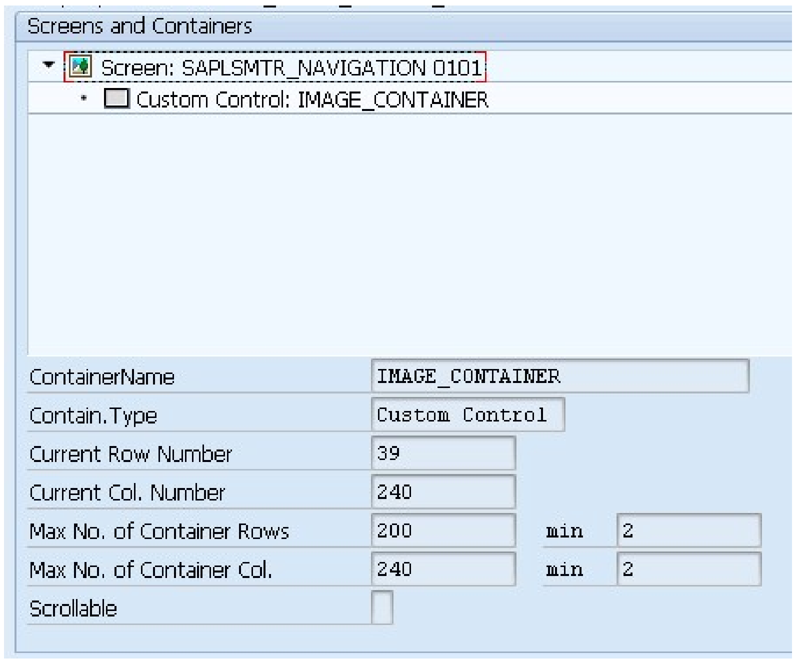


Loaded Programs

C..	P..	P..	Program Name	A..	Header of Program Group	N..	O.	Load Size	Last Chan...	Last Ch...	Index
			SAPLSMTR_NAVIGATION		SAPLSMTR_NAVIGATION		EN	207,872	02/04/2008	20:51:54	0
			SAPMSSYD		SAPLSMTR_NAVIGATION		EN	21,504	11/07/2006	07:31:42	1
			SAPFSYSCALLS		SAPLSMTR_NAVIGATION		EN	7,168	09/09/2004	14:18:32	2
			SAPLSMTR_NAVIGATION_MODULES		SAPLSMTR_NAVIGATION_MODULES		EN	178,176	02/04/2008	20:51:54	3
			SAPLSGUI		SAPLSGUI		EN	83,968	02/04/2008	20:53:54	4
⇨			SAPLPRGN_AUTH		SAPLPRGN_AUTH		EN	91,136	02/04/2008	20:53:08	9
			WDR_TASK==============...		WDR_TASK==============...		EN	12,288	02/04/2008	21:37:04	11

Displays all currently loaded programs and their global variables

Screen Analysis



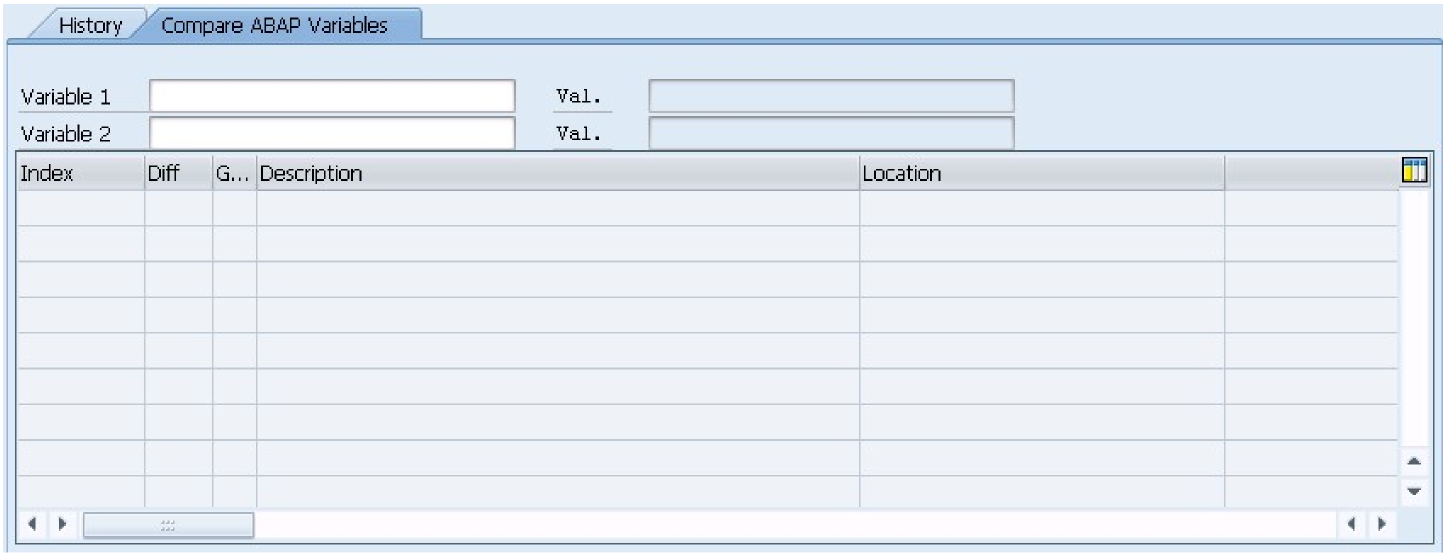
Screens and Containers

- Screen: SAPLSMTR_NAVIGATION 0101
  - Custom Control: IMAGE_CONTAINER

ContainerName	IMAGE_CONTAINER		
Contain.Type	Custom Control		
Current Row Number	39		
Current Col. Number	240		
Max No. of Container Rows	200	min	2
Max No. of Container Col.	240	min	2
Scrollable	☐		

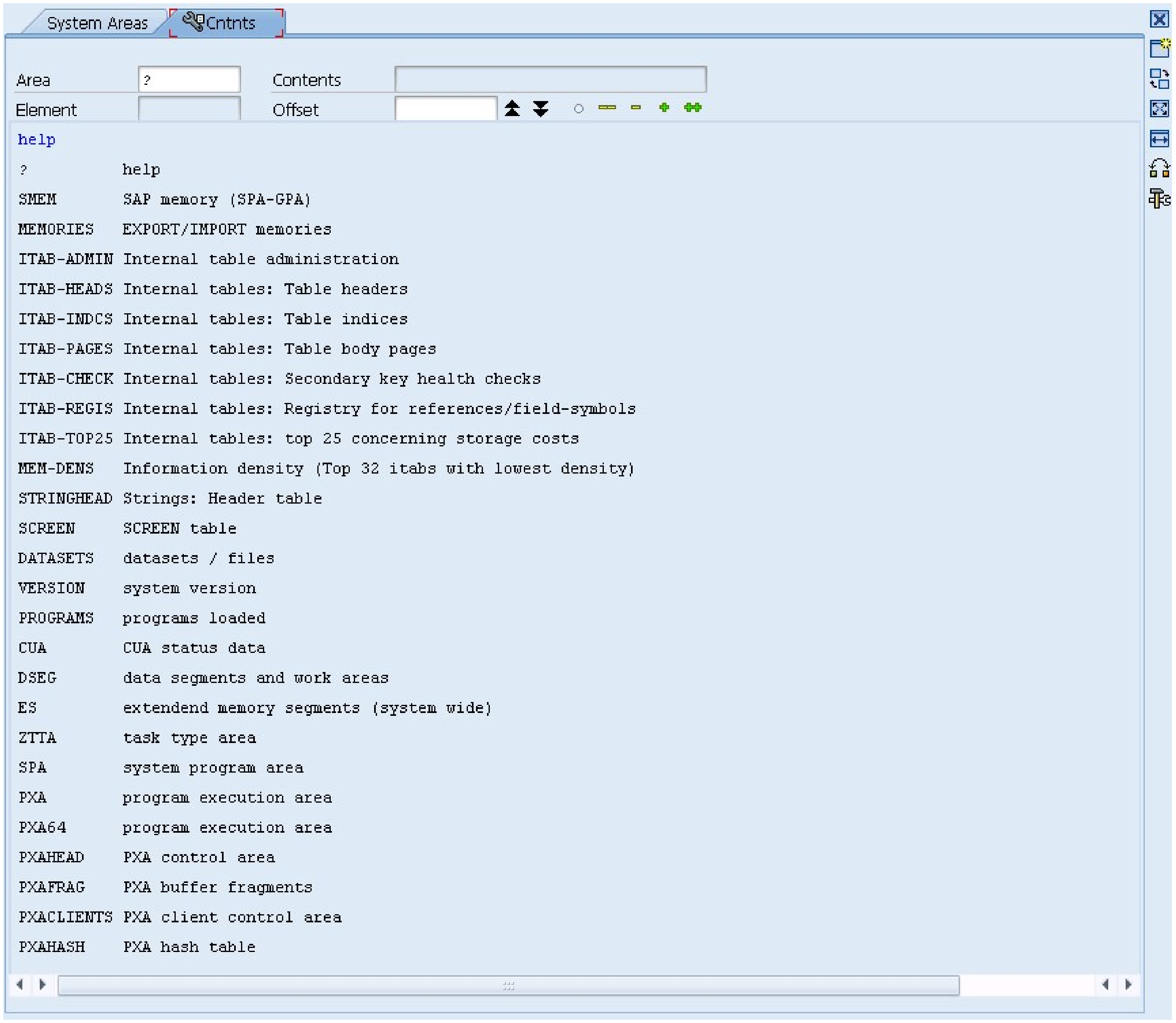
Displays the screen attributes and the subscreen stack

Diff Tool



Allows you to compare to variables looking for the differences between the two. Great for comparing two internal tables

System Areas



Area ?
Contents
Element
Offset

```
help
?          help
SMEM       SAP memory (SPA-GPA)
MEMORIES   EXPORT/IMPORT memories
ITAB-ADMIN Internal table administration
ITAB-HEADS Internal tables: Table headers
ITAB-INDCS Internal tables: Table indices
ITAB-PAGES Internal tables: Table body pages
ITAB-CHECK Internal tables: Secondary key health checks
ITAB-REGIS Internal tables: Registry for references/field-symbols
ITAB-TOP25 Internal tables: top 25 concerning storage costs
MEM-DENS   Information density (Top 32 itabs with lowest density)
STRINGHEAD Strings: Header table
SCREEN     SCREEN table
DATASETS   datasets / files
VERSION    system version
PROGRAMS   programs loaded
CUA        CUA status data
DSEG       data segments and work areas
ES         extendend memory segments (system wide)
ZTTA       task type area
SPA        system program area
PXA        program execution area
PXA64      program execution area
PXAHEAD    PXA control area
PXAFRAG    PXA buffer fragments
PXACLIENTS PXA client control area
PXAHASH    PXA hash table
```

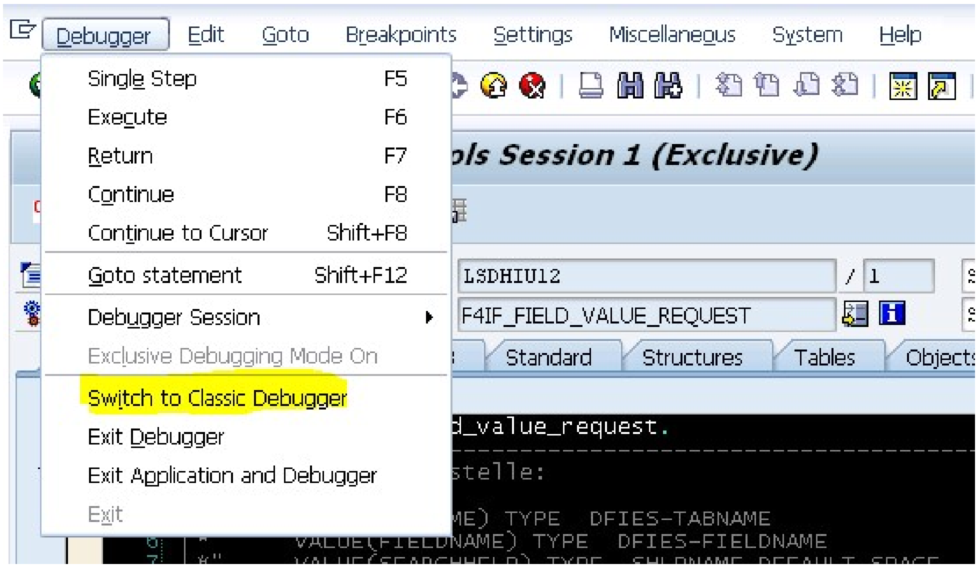
Displays system areas such as SMEM, ABAP Memory, Datasets, Screen Attributes, etc.

<u>Other Debugger Basics</u>

Classic Debugger

Some tools are not available in the new debugger that were available in the classical debugger. In most cases there is a work around.

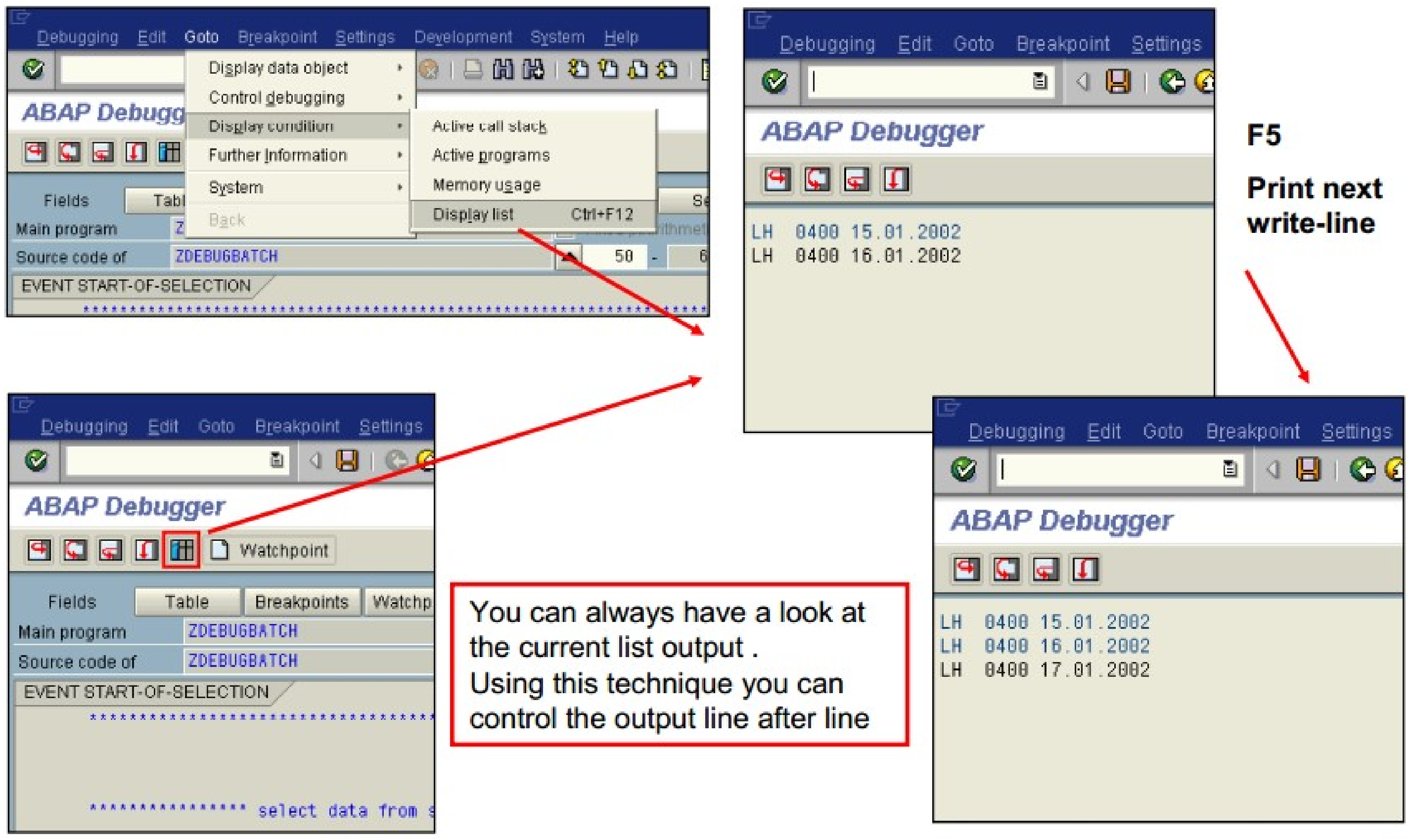
To switch to classical debugger from new debugger (and vice versa) click Switch to Classic Debugger:


Debugger Edit Goto Breakpoints Settings Miscellaneous System Help
Single Step F5
Execute F6
Return F7
Continue F8
Continue to Cursor Shift+F8
Goto statement Shift+F12
Debugger Session
Exclusive Debugging Mode On
Switch to Classic Debugger
Exit Debugger
Exit Application and Debugger
Exit
ols Session 1 (Exclusive)
LSDHIU12
F4IF_FIELD_VALUE_REQUEST
Standard Structures Tables Objects

```
d_value_request.
stelle:
ME) TYPE  DFIES-TABNAME
VALUE(FIELDNAME) TYPE  DFIES-FIELDNAME
```

Display WRITE output in the debugger

In the Classic Debugger there was an option for displaying the output of a write statement. Seen here:



```
LH  0400 15.01.2002
LH  0400 16.01.2002
```

```
LH  0400 15.01.2002
LH  0400 16.01.2002
LH  0400 17.01.2002
```

You can always have a look at the current list output .
Using this technique you can control the output line after line

To display the same data in the **New Debugger**

(SAPMSSY0)%_LIST is an internal table that holds the values for a WRITE statement. More on displaying an internal table

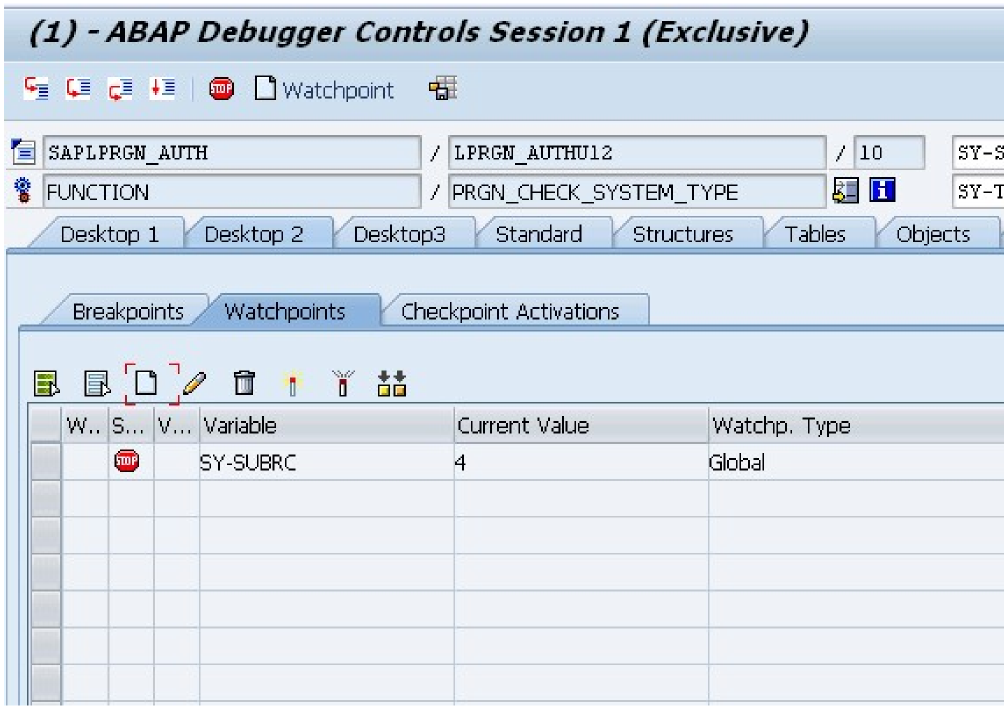
below

Creating Watch Points in the debugger

From any view in the debugger there is an option to click the Watchpoint to create a new watchpoint.

Also from the watchpoint work area the can be clicked to open up.

Here is an example of the watchpoint work area. Both buttons are visible from here.



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

Watchpoint

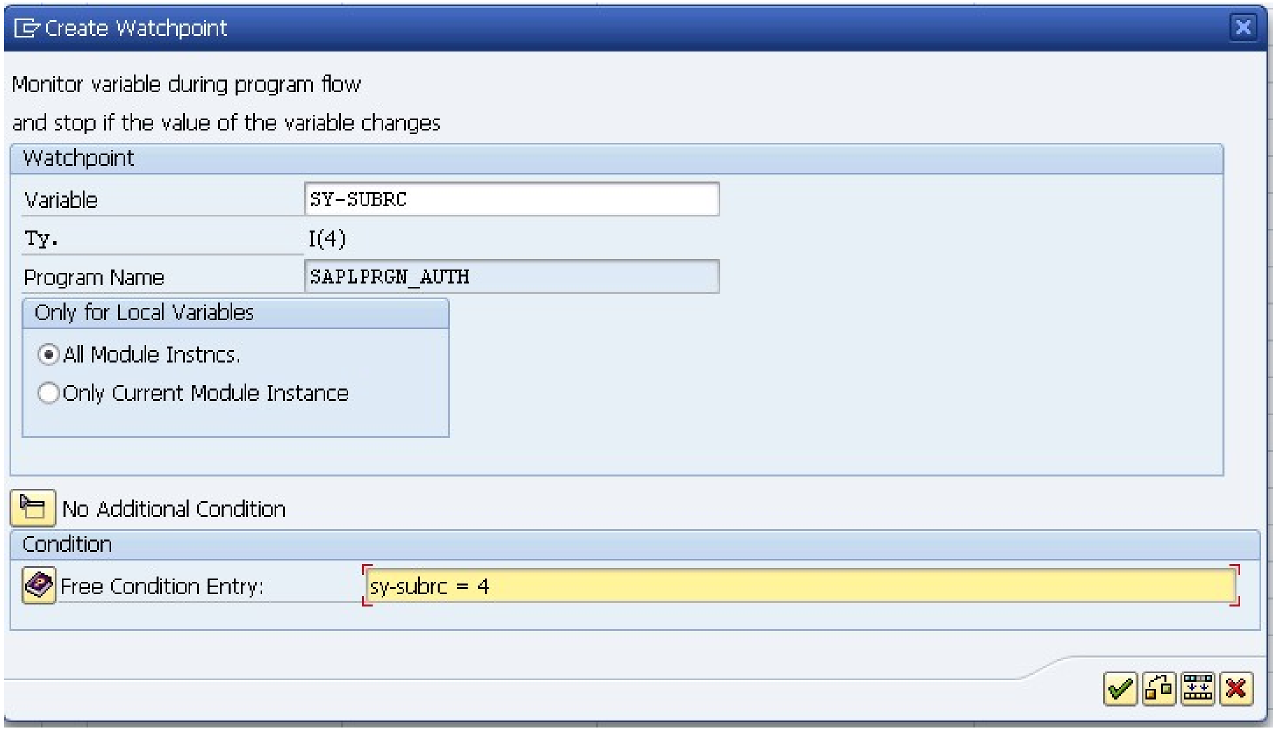
SAPLPRGN_AUTH / LPRGN_AUTHU12 / 10 SY-S
FUNCTION / PRGN_CHECK_SYSTEM_TYPE SY-T

Desktop 1 | Desktop 2 | Desktop3 | Standard | Structures | Tables | Objects

Breakpoints | Watchpoints | Checkpoint Activations

W..	S...	V...	Variable	Current Value	Watchp. Type
	🛑		SY-SUBRC	4	Global

When a new watch point is created the following is displayed.



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

Watchpoint

Variable	SY-SUBRC
Ty.	I(4)
Program Name	SAPLPRGN_AUTH

Only for Local Variables

● All Module Instncs.
○ Only Current Module Instance

No Additional Condition

Condition

Free Condition Entry: sy-subrc = 4

The **variable** is the field we are watching.

Additional conditions can be added at the bottom. In this case we can see that the breakpoint will only execute if sy-subrc = 4. You can also use the AND and OR operators to compare other variables in the condition.

Displaying internal tables in the old debugger:

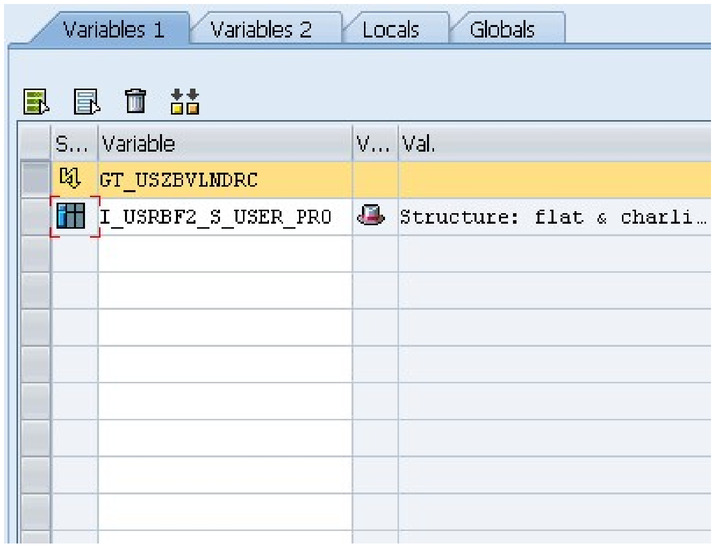
itab - displays the header line

itab[] - displays the table body

***itab[] -** displays the table header

&itab[] - displays the table reference

This is generally obsolete in the new debugger since entering an internal table into the **variable fast display** work area will allow you to look at all of these depending on where you click.



S...	Variable	V...	Val.
	GT_USZBVLNDRC		
	I_USRBF2_S_USER_PRO		Structure: flat & charli…

- shows the table body

Double clicking the name of the table displays the header

(note: in this screen shot the means that the variable is not known. If you see this then the variable is most likely out of scope)

Debugging in client production system (very poorly put together in the pdf...)

There is no 100% safe way to debug in a production system. Data inconsistencies can be created if proper steps are not taken.

When switching screens database commits may occur. But if you try to rollback other database commits the ones from the previous screen will still be committed. So you are responsible for keeping up with data consistency. Another case might be if functions are called **IN UDPATE TASK** then you might roll these back before the commit work but other direct commits will already be in place.

Also, if a lock is active on a table while you are debugging this will lock other users out of the table.

Start Debugging (whether methods above or through SM50)

Check in SM50 if the process is in debug mode

If not leave debugging and try again later

If yes tehn debugging is possible

If you stop (abort) debugging choose DB rollback from menu

Leave debugging.