The New ABAP Debugger (NetWeaver 2004s)

Christoph Stoeck
Boris Gebhardt

THE BEST-RUN BUSINESSES RUN SAP™

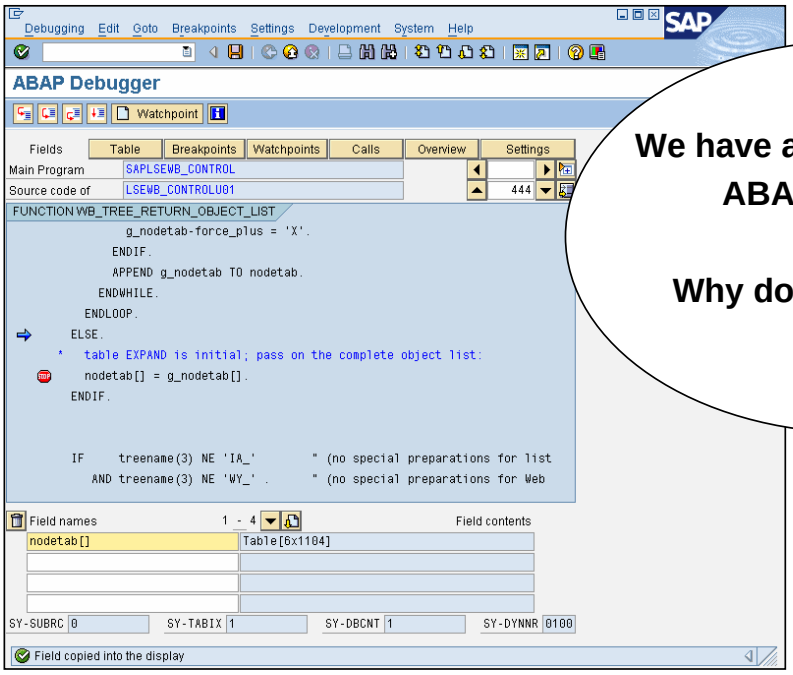
ABAP Debugger: Content

- Motivation
- Architecture & Handling
- UI Basics
- New ABAP Frontend Editor
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints

ABAP Debugger: Content

- **Motivation**
- Architecture & Handling
- UI Basics
- New ABAP Frontend Editor
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints

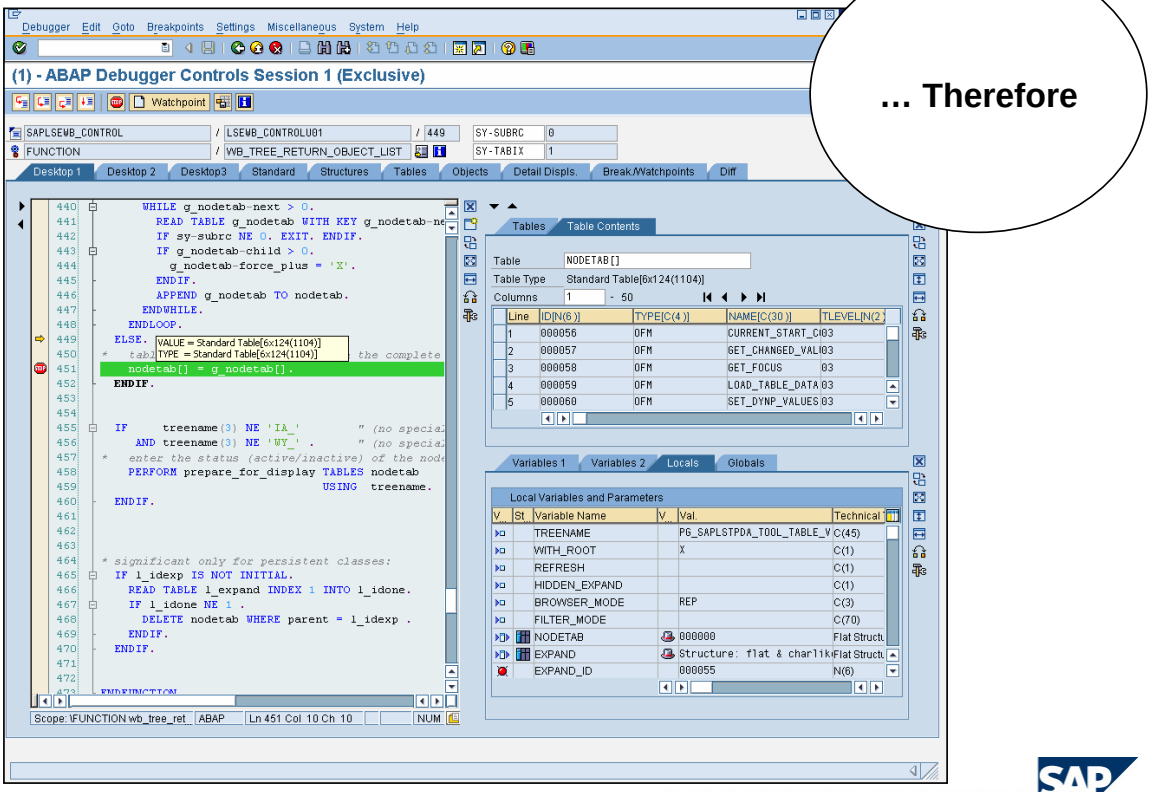
Motivation – The Classic Debugger



We have already a powerful ABAP debugger.

Why do we need a new one ??

Motivation – The New ABAP Debugger



... Therefore

THE BEST-RUN BUSINESSES RUN SAP™

- The first version of the New ABAP Debugger is available with NetWeaver04. The full range of features described in the presentation is available with NetWeaver 2004s.

Current Status – Classic Debugger

Classic Debugger

Technology

- Debugger and debuggee run in the same (internal) session
- Debugger dynpros placed “in-between”

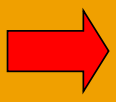
Consequences

- Not all ABAP code can be debugged (no RPERFs: Conversion / Field exit))
- Not free of side effects (F1, F4 help, list output)
- Implementation of new features not always straight-forward
- No chance to use modern UI techniques (no ABAP allowed in the debugger !)

Goals – New ABAP Debugger

Higher productivity for development & support using ABAP debugger

- More robust debugger architecture (no side effects)
- Possibility to implement new features (e.g. a diff tool for internal tables) faster and with less risks
- More flexible & extensible state-of-the-art debugger UI



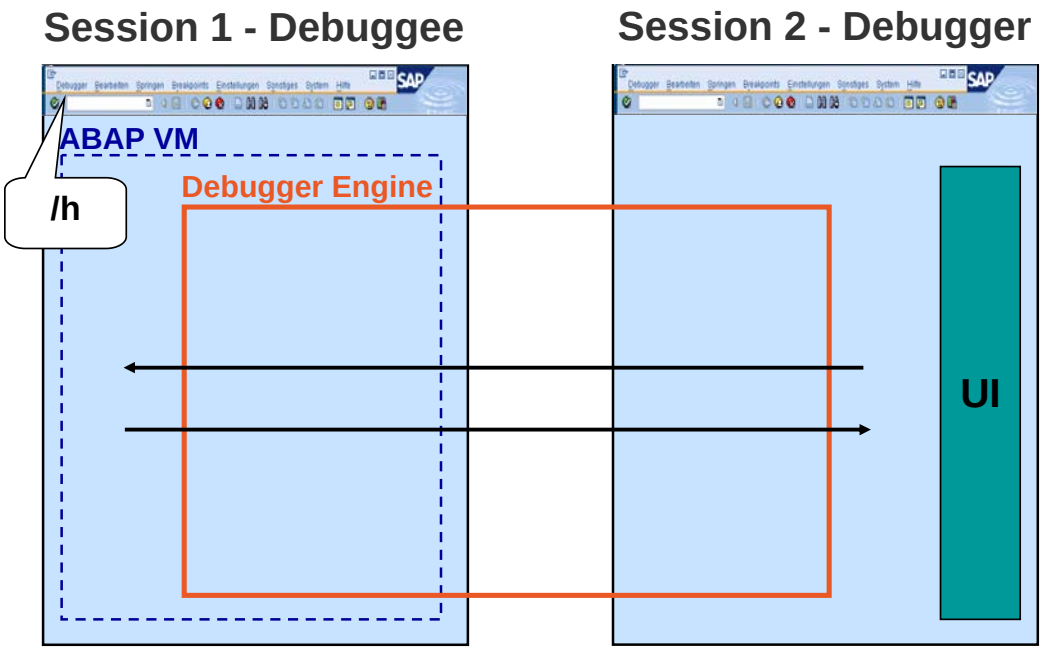
Use two separated sessions for the debugger and the application

Content

- Motivation
- **Architecture & Handling**
- UI Basics
- New ABAP Frontend Editor
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints

Two Process Architecture

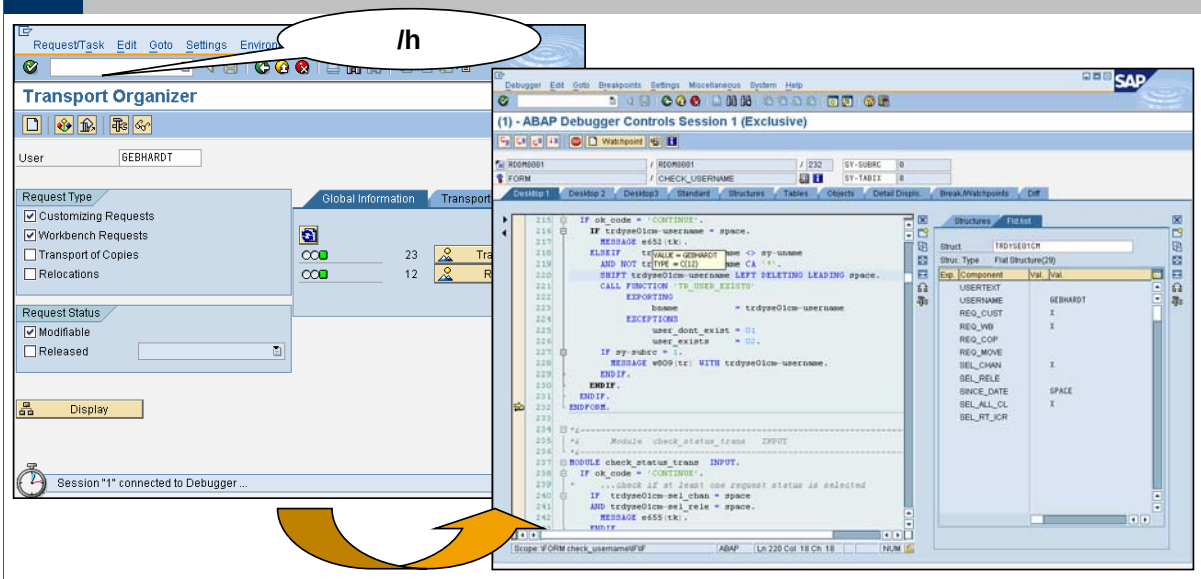
The New Debugger is attached to an “external session”



The classic Debugger runs in the same internal session as the application itself. Therefore after entering a new roll area (submit/call transaction) the debugger is restarted-> all debugger breakpoints are gone and the UI is refreshed. (You have to type in your variables again…).

The new ABAP Debugger is totally separated from the application and is attached to the external session and not the internal session. Therefore you can debug roll area (internal session) switches without any disturbance.

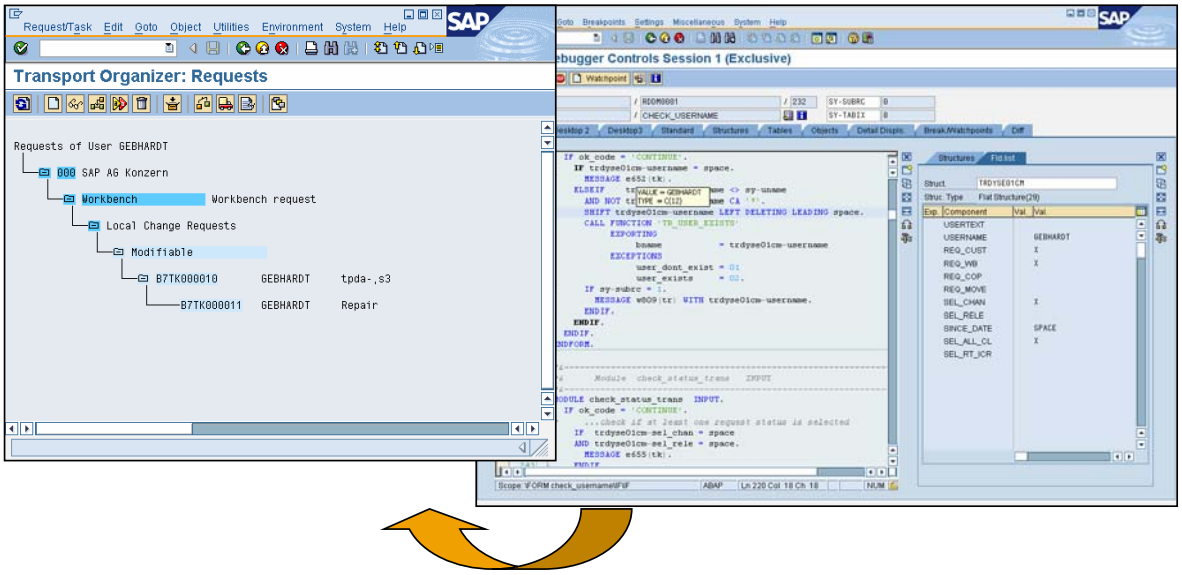
Two Process Architecture - Consequences



- **The New Debugger is started in a separated session, after prompting "/h"**
- **The debuggee is inactive while the debugger is active.**

Advantage: **During debugging you still see your last screen input**

Two Process Architecture - Consequences



- **The debugger is still available but inactive, when the program finished**
- **The debugger is not closed as long as the debuggee session is alive !**
- **You may detach the debugger by prompting “/hx” in the debuggee session**

Advantage: **The debugger with all your settings, variables, breakpoints,… is always available, when you restart debugging !**

- As the New ABAP Debugger is attached to external session the debugger is not closed, when the application finished.
- Only when you close the screen or if you use ok-code /hx the debugger is detached.
- You may use the debugger setting (Menu: Setting ->Display/Change debugger Settings): “Close Debugger After 'F8' and Roll Area End “ to influence this behavior.
  If you switch on this setting, then the Debugger will be closed automatically after pressing “Continue” (F8) and the end of the roll area. This is more or less the behavior of the Classic
  Debugger, and in order to make the migration smooth this is the default setting.

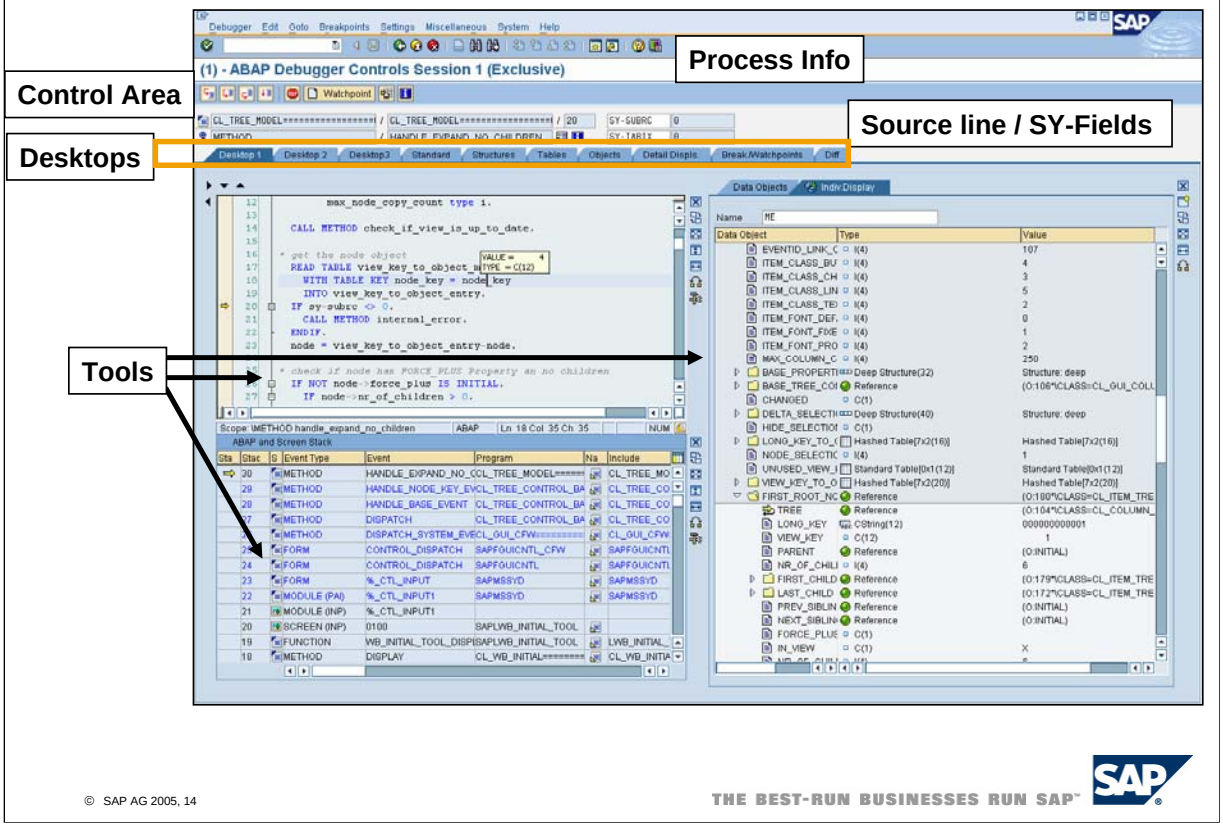
Both debugger: differences in handling

	Classic Debugger	New Debugger
Subject that is being debugged	Internal session (roll area) ■ (breakpoints) ■ UI settings ■ tool content, e.g. variables lost in new roll area	External session (GUI window) ■ breakpoints ■ UI settings ■ tool content, e.g. variables NOT lost -> /hx closes debugger -> option: close automatically
Window where the debugger is presented in	Same window like application	Separate window ■ last input screen visible ■ focus follows active window

Content

- Motivation
- Architecture & Handling
- **UI Basics**
- New ABAP Frontend Editor
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints

New Debugger UI – Main Parts



- The process info contains:
  - ◆The session number you are attached to
  - ◆The debug mode (/hs-> system debuging active/ RFC / HTTP/ATTACHED etc.)
  - ◆Info: Is this an exclusive or non-exclusive debug session
    In a non-exclusive debug session each Rollin/Rollout e.g. during a debug step is connected to a db commit
    In an exclusive debug session the current work process is locked for your debug session, therefore there is no need for a db commit during rollout/rollin, because you will always re-enter the same locked work process.
    The system always tries to get an exclusive debug session when you start debugging (in a productive client only exclusive debug sessions are allowed!)
    But the number of exclusive debug session is limited by the profile parameter rdisp/wpdbug_max_no, because if you would lock all dialog work processes with an exclusive debug session then nobody would be able to logon …
    In a development system we recommend rdisp/wpdbug_max_no = number of dialog processes / 2.
    All this is valid for the Classis and the New ABAP Debugger !

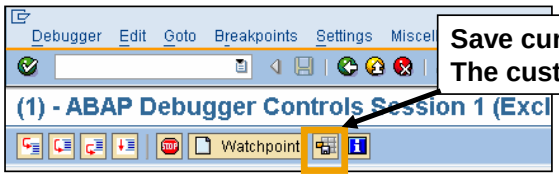
New Debugger UI – Desktops

User specific desktops

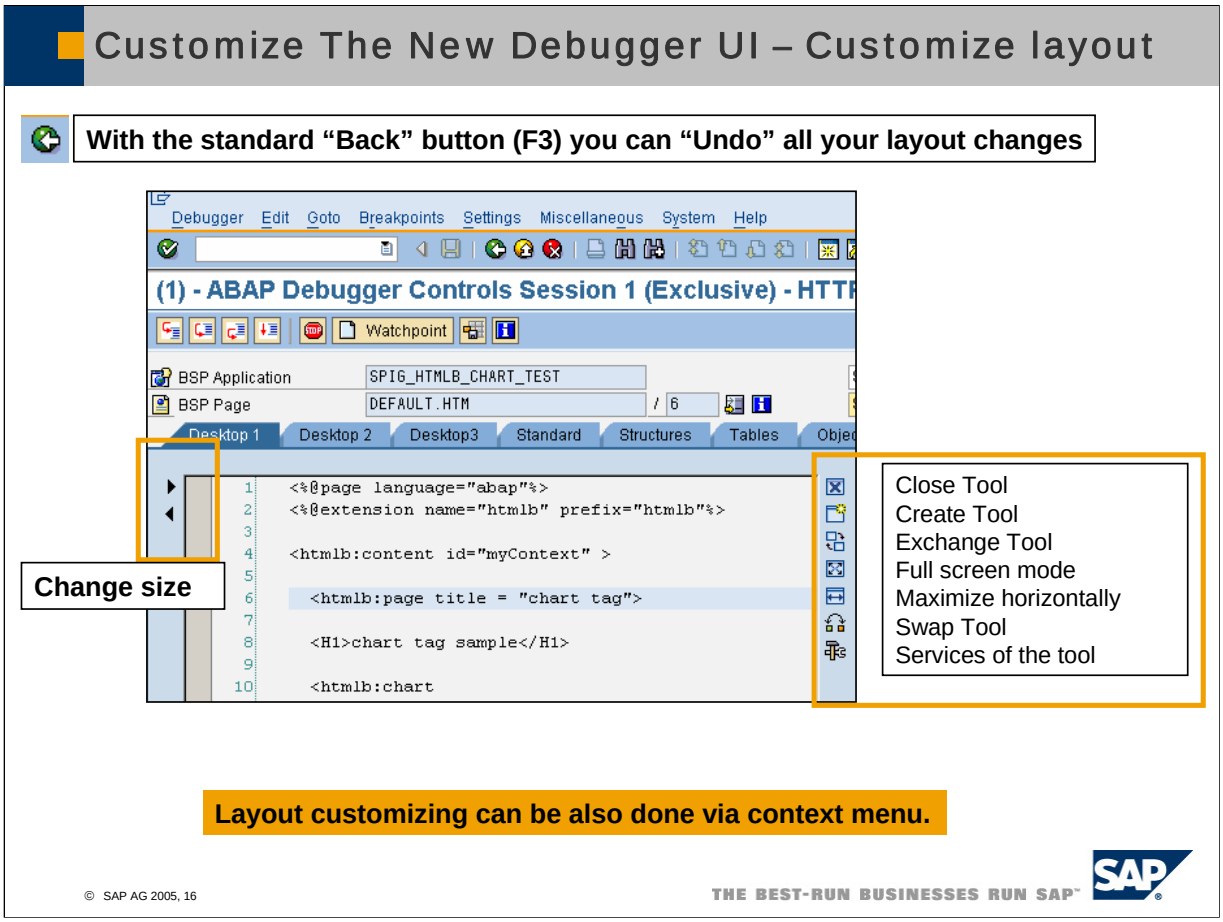
Standard desktops

The New Debugger provides:

- **Three user specific desktops, which you can customize and save as your favorite debugger environment**
- **Seven standard desktops, which should cover most of the common working conditions in the debugger:**
  - **Standard:** **Stepping through the code (Editor, Stack, Quick Watch)**
  - **Structures:** **Display structures**
  - **Tables:** **Display tables**
  - **Objects:** **Display objects**
  - **DetailDispls:** **Display strings , simple fields …**
  - **Break-/Watchpoints:** **Maintain your break-/watch-/checkpoints**
  - **Diff** **Compare Variables**



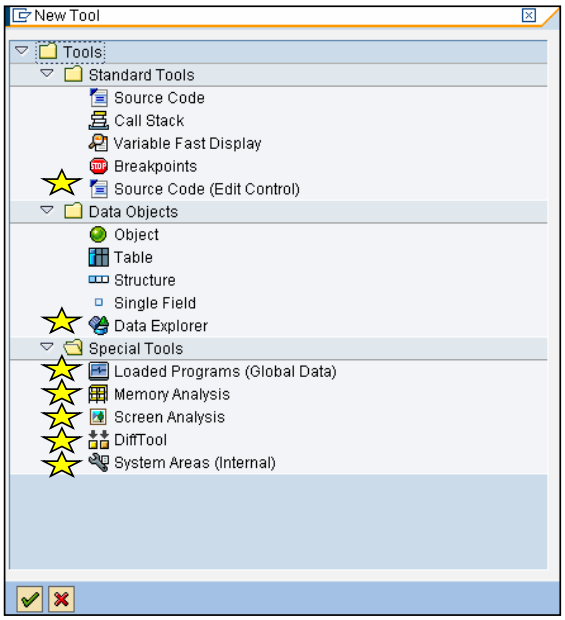
Save current layout of the user specific desktops.
The customizing of the standard desktops is NOT saved !



- The New ABAP Debugger (version NetWeaver 2004s) allows debugging of HTTP sessions as well. In this screenshot you see the debugging of a Business Server Page (BSP).

Customize The New Debugger UI – Tools

Available Tools:
★ **New with NetWeaver 2004s**

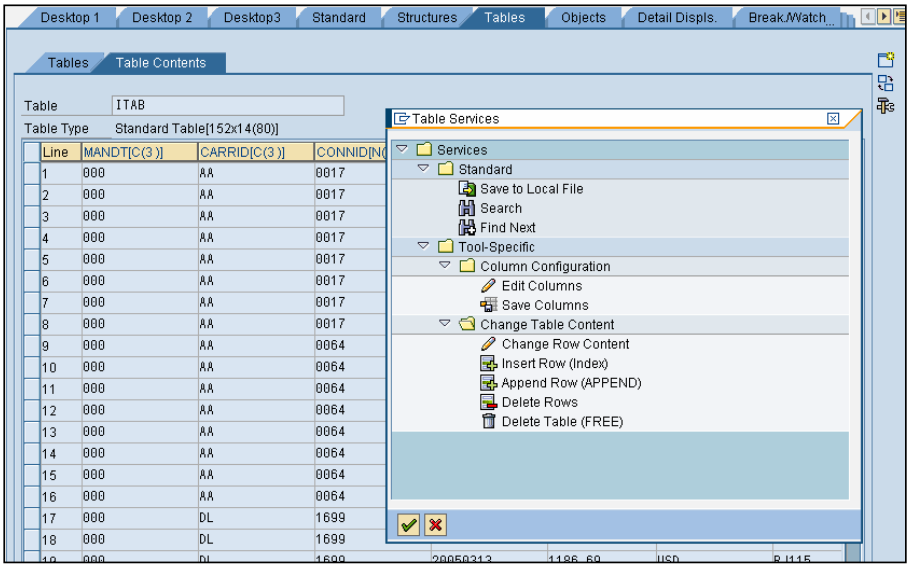


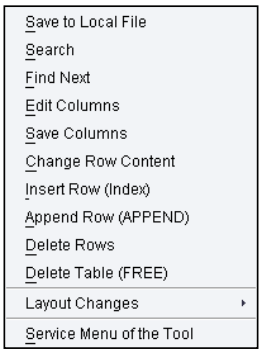
You may create up to 4 instances of one tool on one desktop. (e.g. to compare 4 internal tables)

- **Standard Debugger Tools:**
  - (Tools, every debugger should have)
  - Source Code Display (Current source code extract)
  - Stack Display (Current ABAP & Screen stack)
  - Variable Fast Display (Quick watch of variables)
  - Breakpoint Display (Maintain your breakpoints)
- **Detail displays for variables:**
  (Tools, which are specialized for the analysis of one dedicated ABAP data type)
  - Internal Table Display (Internal Tables)
  - Object Display (Objects and classes)
  - Structure Display (Structures)
  - Single Field Detail Display (Simple data types, like C, N, D, T, STRING, …)
  - Data Explorer (Generic and tree like variable display)
- **Special tools:**
  - Diff Tool (Compare variables)
  - Memory Inspector (Integrated Memory Inspector[1])
  - Loaded Programs (Attributes of all loaded programs and their

Customize The New Debugger UI – Tool Services

All features of a tool can be accessed via the tool service button…





…or the context menu

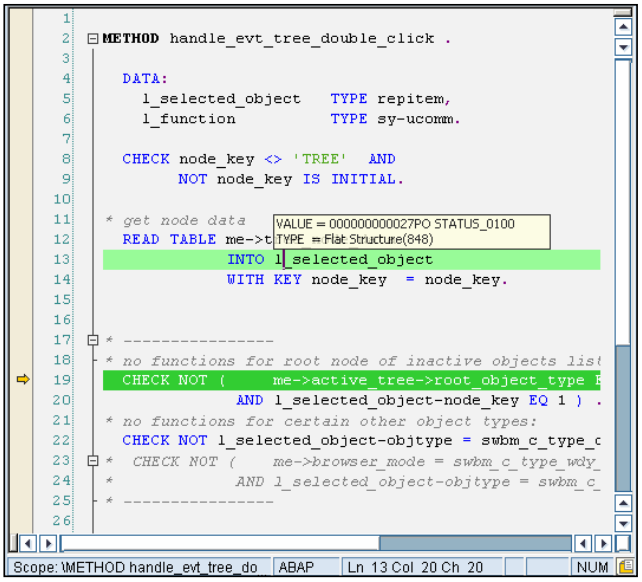
THE BEST-RUN BUSINESSES RUN SAP™

- In order to make it easy finding all the features of the different tools, each tool uses a menu tree (the so called tool services) .
- The tool services are separated in standard services (download, search) which are provided by more or less all tools and the tool specific services (e.g. for the table view: Change row content)
- By the way, in the New ABAP Debugger you can change the content of internal tables in a very convenient way:
  - Insert/Append using a template (e.g. insert a copy of table line 11 and I will do only some changes before the real insert)
  - Free of an internal table

Content

- Motivation
- Architecture & Handling
- New ABAP Debugger UI Basics
- **New ABAP Frontend Editor**
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints

ABAP Frontend Editor



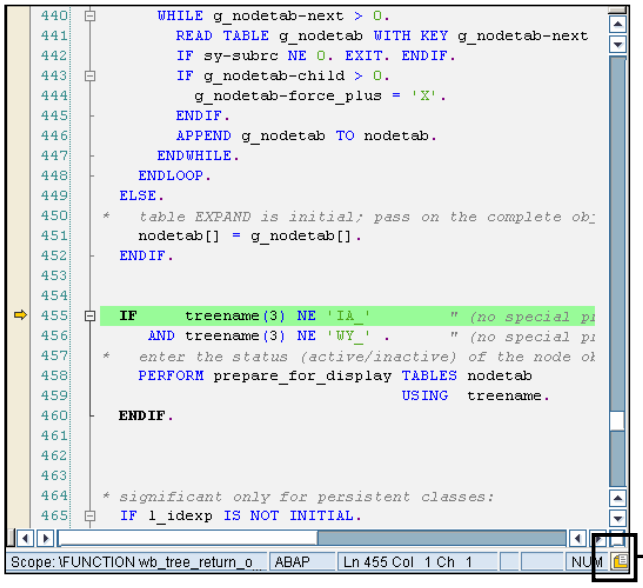
Main features

- **Syntax coloring**
- **Block marking / folding**
- **Display variable content & type**
- **Easy BP handling**

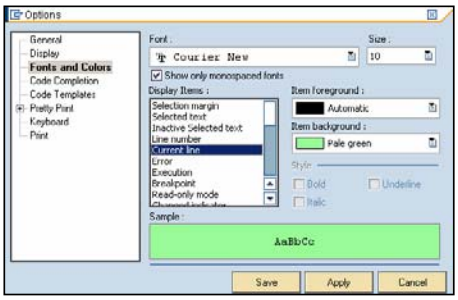
New ABAP Frontend Editor is available with NetWeaver 2004s plus 6.40 GUI (PL10).

- Please check SDN (https://www.sdn.sap.com) for a detailed descript of the New ABAP Editor ("**Inside the New ABAP Editor**")

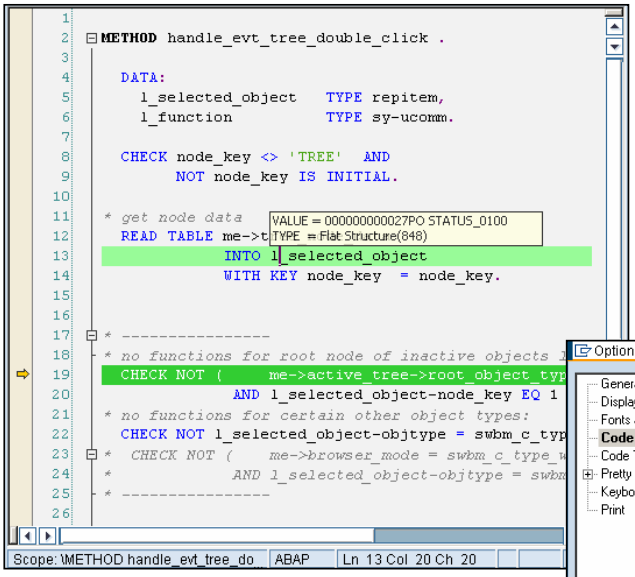
ABAP Frontend Editor – Source Display features



- **Syntax coloring**
- **Block marking / folding**
- **Display scope**
- **Convenient vertical and horizontal scrolling**
- **Powerful search functionality**
- **Control (layout, colors, …) can be customized**



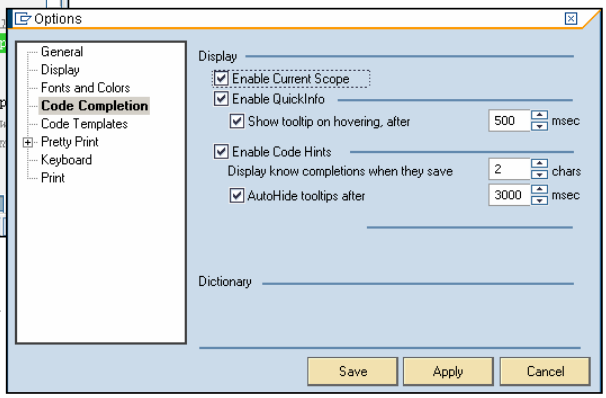
ABAP Frontend Editor – Display Variable Content



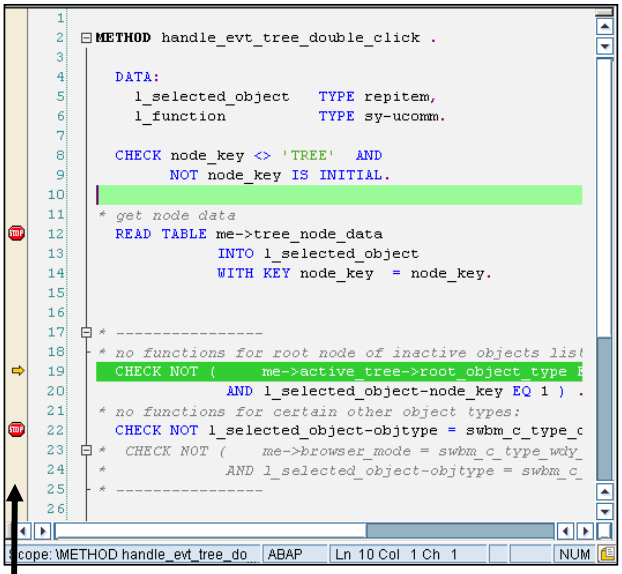
Mouse pointer hovers over variable:

-> **Variable content and type is displayed in a quick info window.**

In the control option dialog you can disable the quick info or adapt the reaction time.



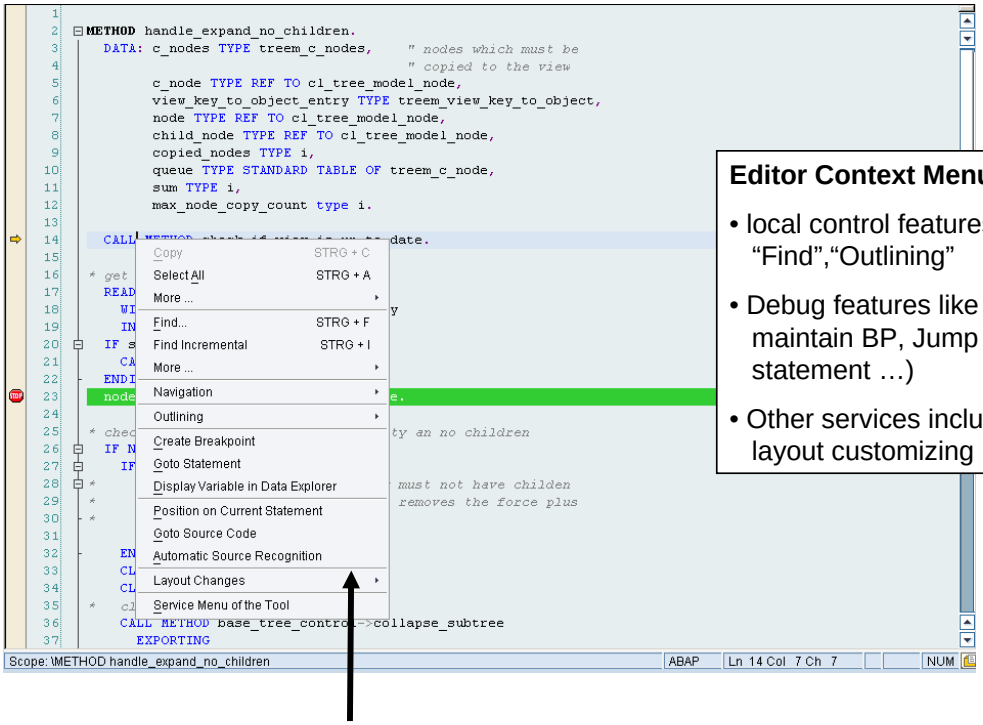
ABAP Frontend Editor – Set Breakpoints



Set / Delete breakpoints by clicking in the left (breakpoint) column .

Double-Clicking in the code will no longer set or delete breakpoints. Deactivate /Activate the breakpoint via context menu.

ABAP Frontend Editor – Context Menu

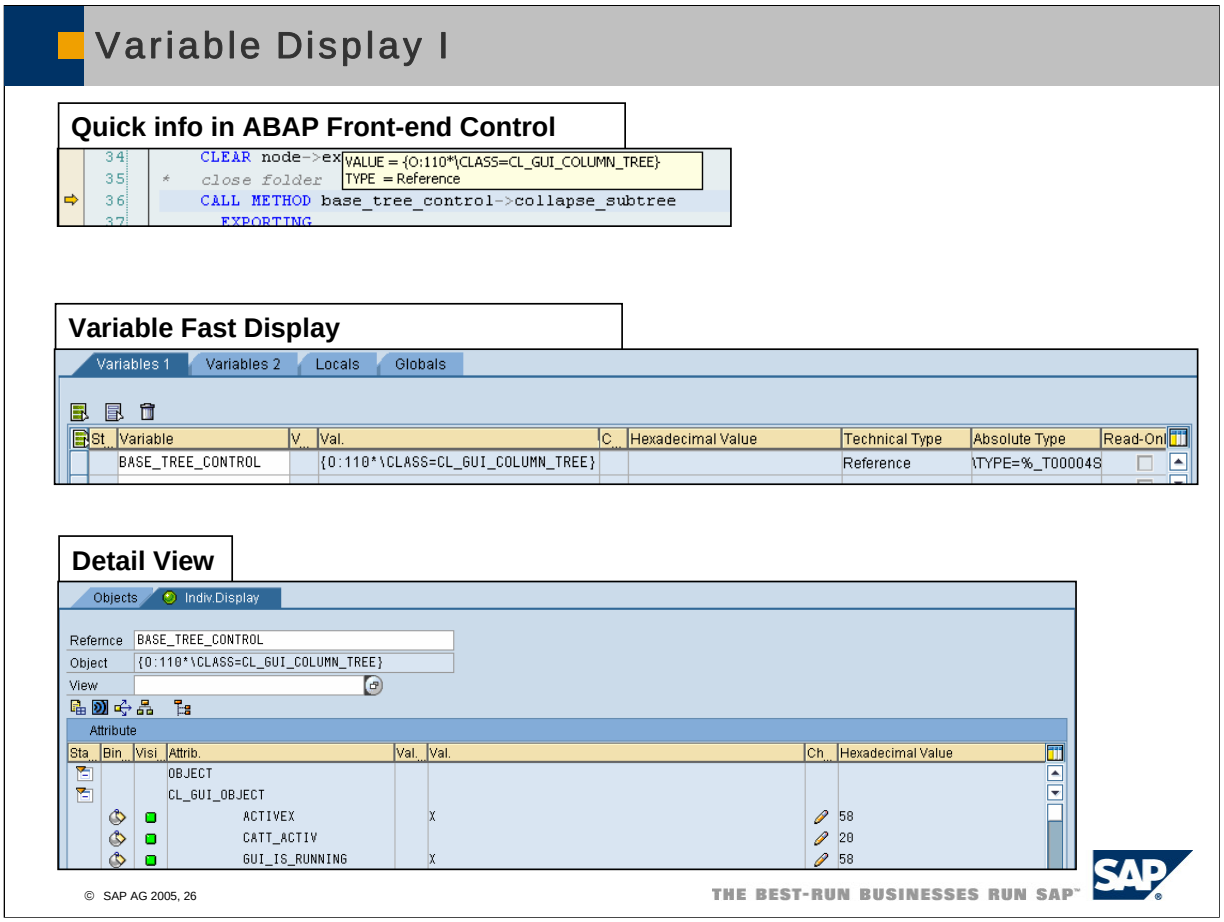


Editor Context Menu contains:

- local control features like “Find”,“Outlining”
- Debug features like maintain BP, Jump to statement …)
- Other services including layout customizing

Content

- Motivation
- Architecture & Handling
- UI Basics
- New ABAP Frontend Editor
- **Variable Display & Navigation**
- Comparing Variables (Diff-Tool)
- Breakpoints & Watchpoints



- The New ABAP Debugger allows analyzing variables on different detail level:

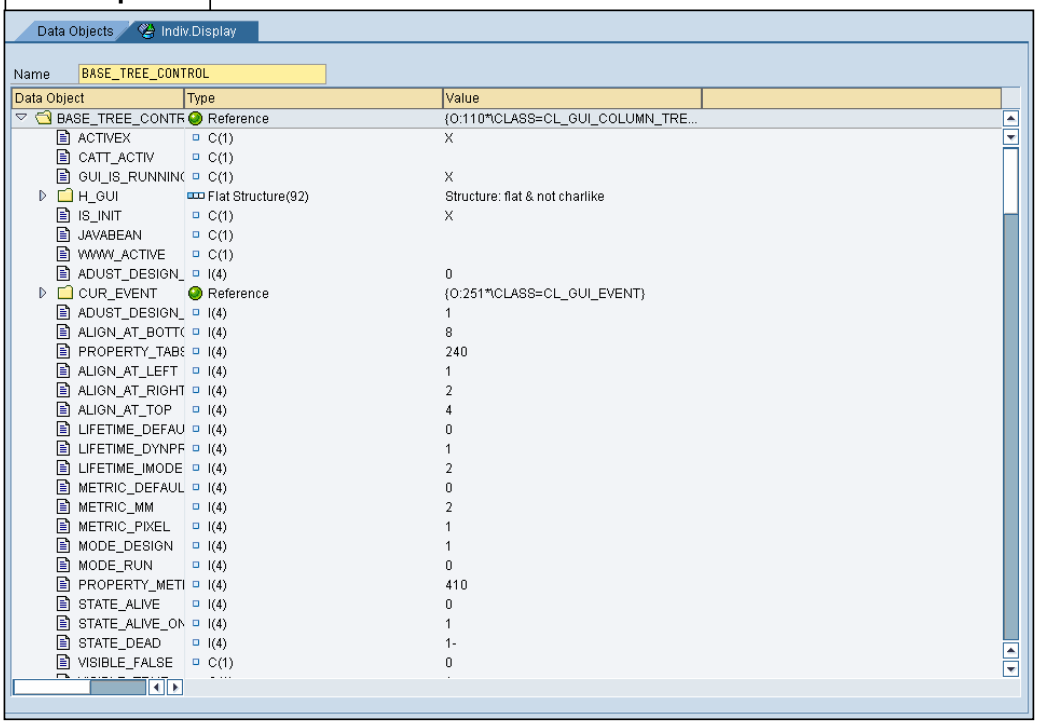
1. While stepping through the code you can just use the quick info to get the value and the type
2. A double-click on the variable name leads to the variable fast display where you get some extra info (e.g. the hex value, absolute type (how is the variable declared), is the variable read only ?
3. A double-click on the variable name in the variable fast display leads to an specialised detail view (e.g. for an object the object view).
   In the detail views you can again double-click on attributes, components in order to navigate to the subcomponents

By the way, the object view provides the possibility to display:

- The full inheritance relationship (including interfaces)
- Events and registered event handler
- Where used list (who points to this instance)
- …

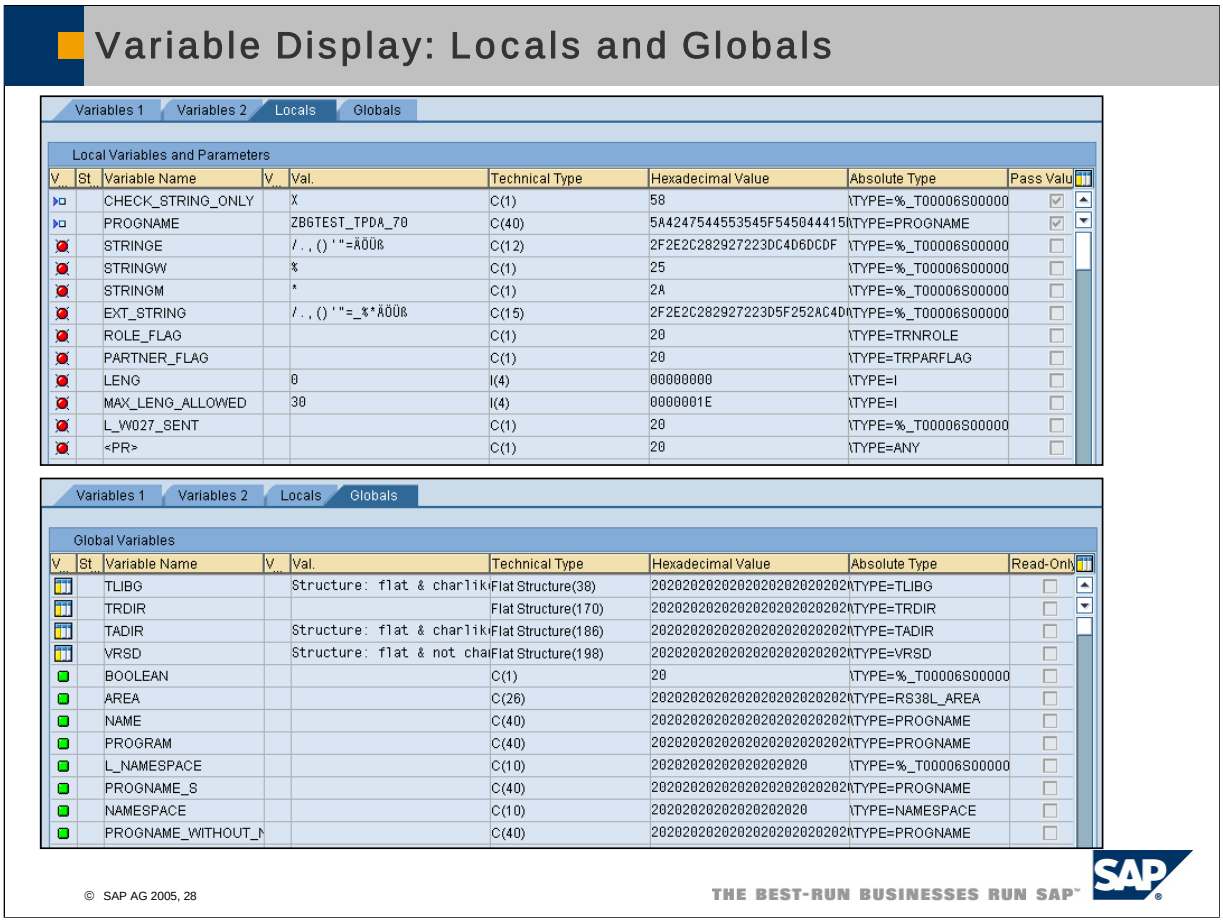
Variable Display II

Data Explorer



- For nested objects, structures, references the data explorer allows you to dive into the depth without loosing the context of the upper levels.

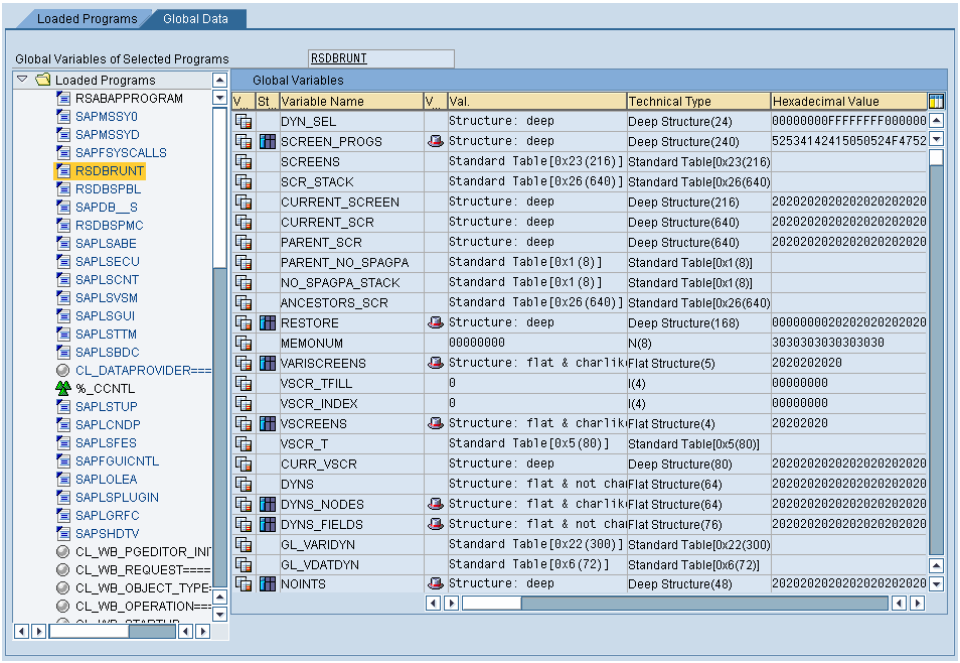
Variable Display: Locals and Globals



- Locals: You get all parameters (including parameter kind) and local variables of the current module – of course with the full navigation power
- Globals: All global variables of the current program

Loaded Programs - Globals

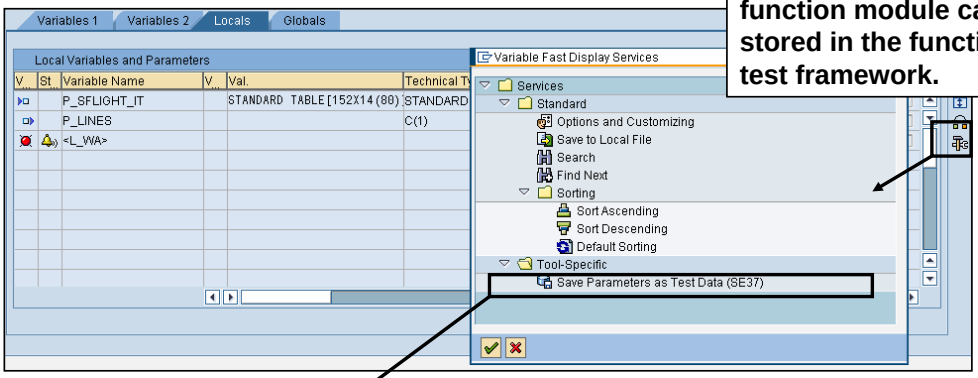
Display all global variables of the loaded programs



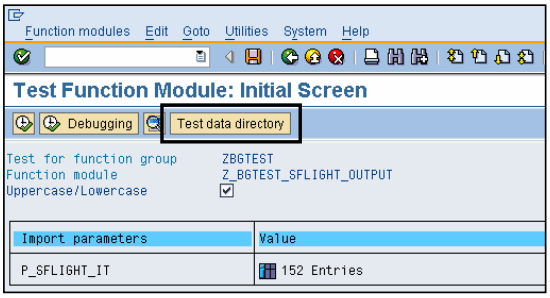
- Two sections:
  Loaded Programs: Here you see the attributes of all programs and the program groups
  Global Data : Get the global variables of each global variable just by double-clicking in the navigation tree.

Download Parameters To Function Builder

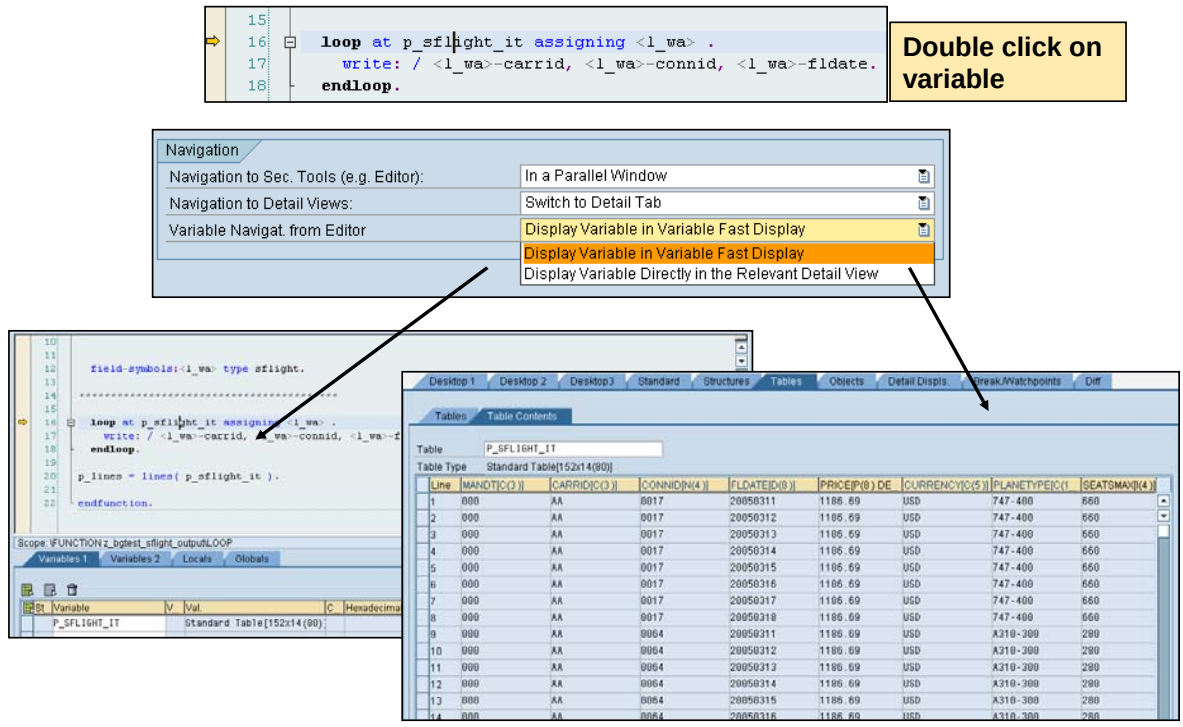
The current parameters of a function module can be stored in the function builder test framework.



Function builder test data dictionary

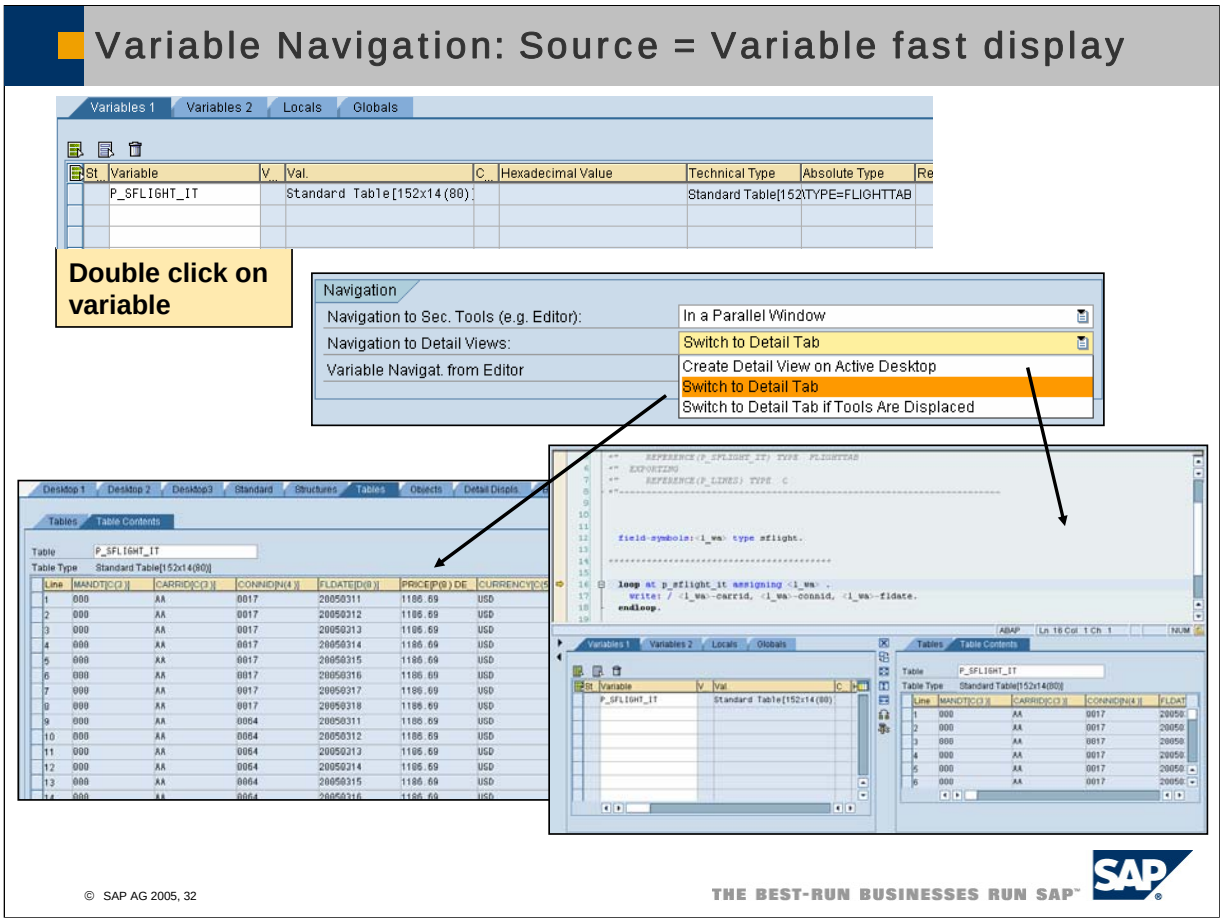


Variable Navigation: Source = Editor



- If you use the new ABAP Editor then the quick info provides already the value of a variable.
  Therefore if you want to get more details then in most cases (and especially if you run the editor in full screen mode) you want to navigate to the detail view instead of the variable fast display which will not deliver
  much more info.
  In this case you should use the option: 'Variable Navigat. from Editor: Display Variable Directly in the Relevant Detail View.
- As all options, you can save your settings so that with the next debugger start these settings are always active.

Variable Navigation: Source = Variable fast display

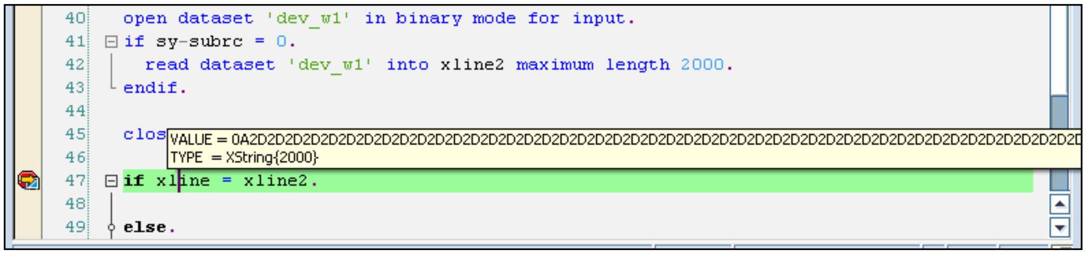


- If you double click on a variable in the variable fast display then per default you navigate to the tab which hosts the appropriate detail view (e.g. table view)
- If you prefer to create the detail view on the current desktop please choose: Navigation to Detail Views: Create Detail View on Active Desktop.

Content

- Motivation
- Architecture & Handling
- UI Basics
- New ABAP Frontend Editor
- Variable Display & Navigation
- **Comparing Variables (Diff-Tool)**
- Breakpoints & Watchpoints

Diff Tool



Why is XLINE <> XLINE2?
Find the difference !!
(XLINE and XLINE2 are only 2000 bytes long …)

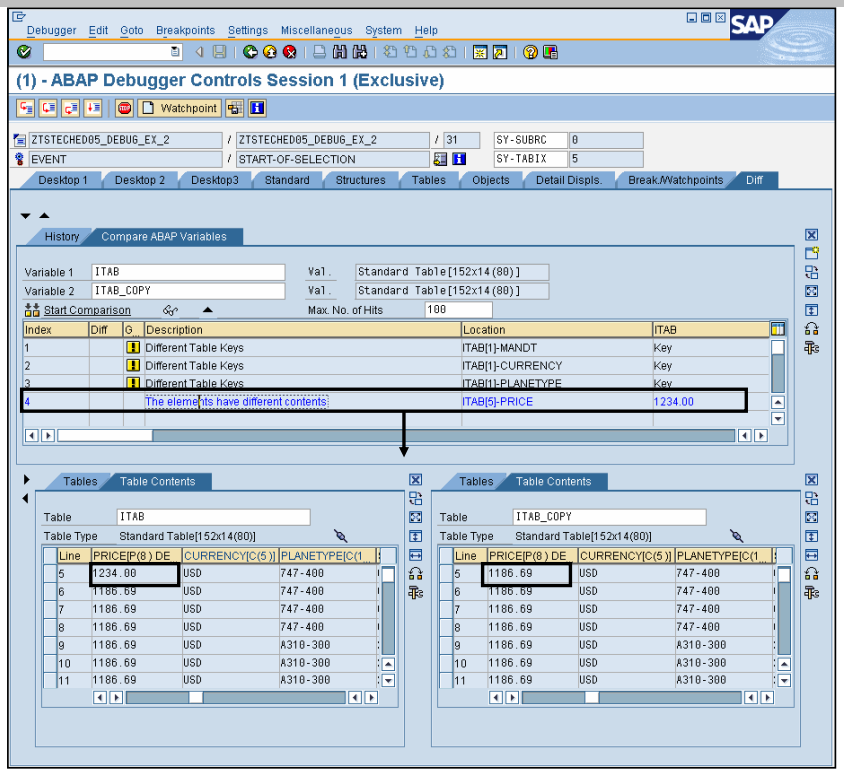
Use the brand new debugger Diff tool !

Diff tool: Compare two compatible ABAP variables:
e.g. two tables, objects, structures, strings,…

Diff tool provides differences concerning type and value.
For value differences the ABAP IF semantic is used.

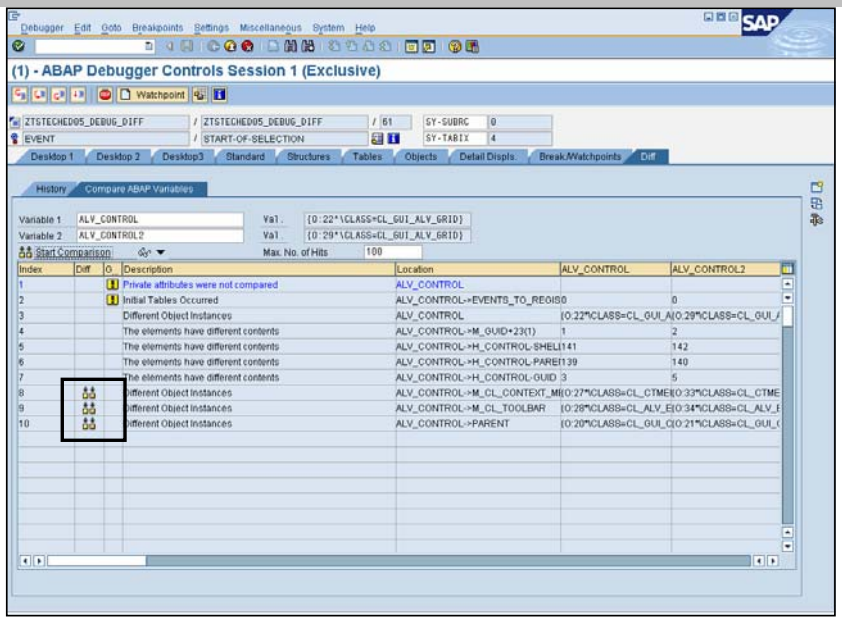
- With the new DIFF tool integrated in the New ABAP Debugger you can compare all variables which can be used in a IF statement as well.
  (e.g. you may compare two structures or two internal tables)
- The DIFF tool provides differences concerning the type (e.g. different component names/types in a structure) and differences concerning values

Diff Tool – Navigation To Connected Detail Views



◆Just by double-clicking on a hit in the diff result list the variables are displayed in parallel and the debugger navigates to this difference.

Diff Tool – Start New Diff For Subcomponents



For subcomponents of type reference or internal table no “deep diff” is executed.
But in the corresponding result lines you find again a diff button to start a comparison for these sub components.
You may use the history tab to get back to the first result list.

- In order to assure a good performance of the diff tool even for deeply nested variables we execute no deep diff.
- This means if you have e.g. an internal table in a structure then we only check if the tables of both structures are equal but you will not get details where they differ.
  To get these details you can run a diff for these subcomponents.

Content

- Motivation
- Architecture & Handling
- UI Basics
- New ABAP Frontend Editor
- Variable Display & Navigation
- Comparing Variables (Diff-Tool)
- **Breakpoints & Watchpoints**

Breakpoints

How many different breakpoint kinds do exist ?

Debugger breakpoints
- **scope: debugging session**
- **set in : debugger**

Perhaps 4. type:

```
BREAK <User-Name>.
BREAK-POINT.
```

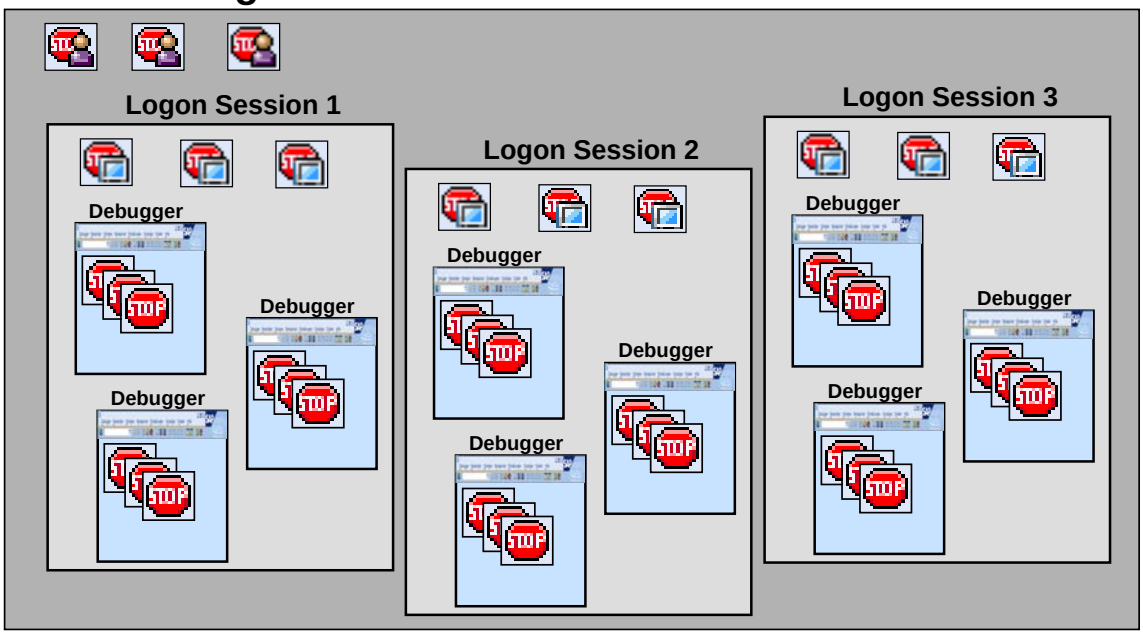
Session breakpoints
- **scope: logon session**
- **set in : development workbench**

User (external) breakpoints
- **scope: all logon sessions (for one user on one server)**
- **set in : development workbench**

- In the classic debugger Debugger breakpoints are only available in the current internal session. By saving the breakpoints they were converted to Session beakpoints.
- In the New ABAP Debugger the Debugger breakpoints are available for all internal sessions of the current external session.
- In the classic debugger User(External) breakpoints can only be used for RFC and http sessions. In the New ABAP Debugger you can use User(External) for all kind of process types.

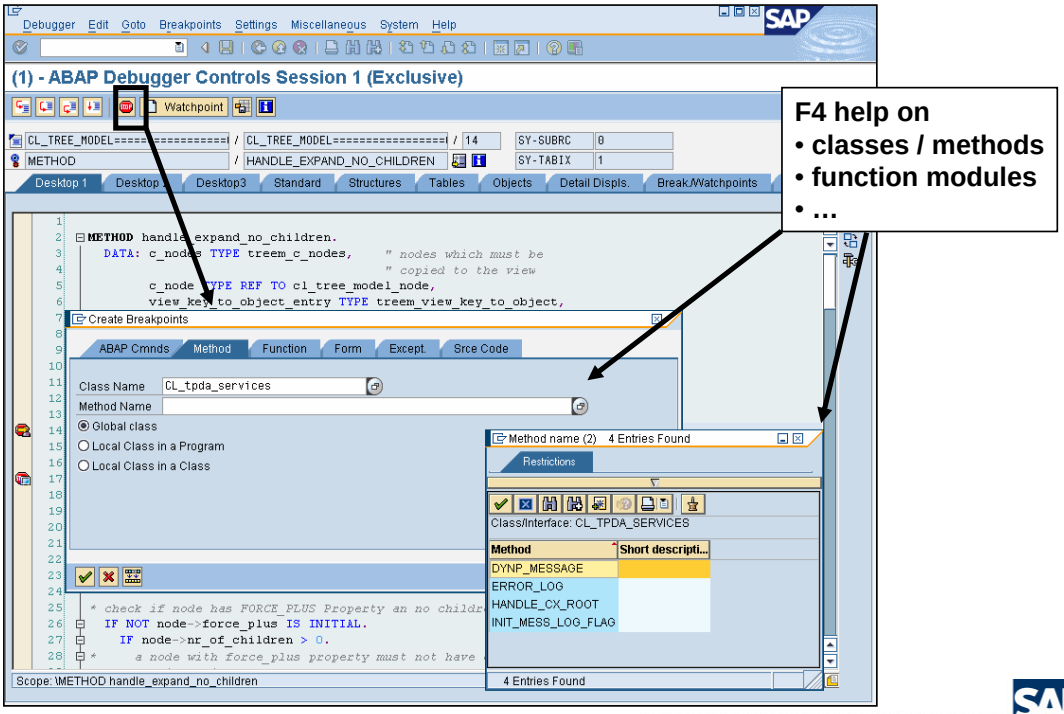
Breakpoints

All logon sessions of one user on one server

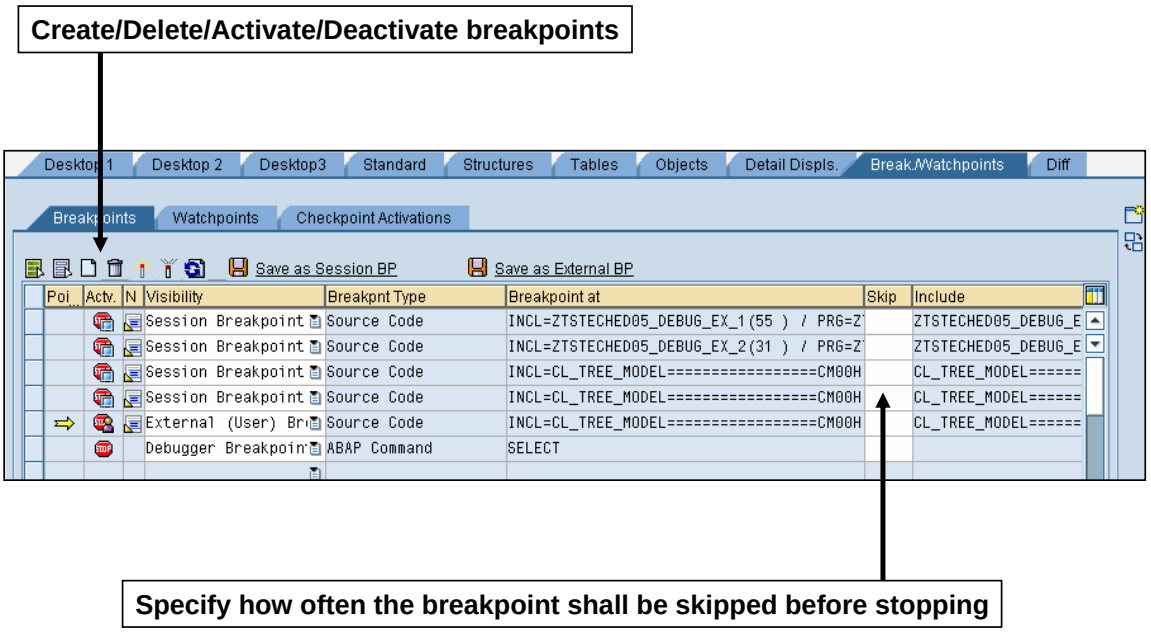


Breakpoints – Create A Breakpoint

Create/Delete a breakpoint in the Editor by clicking on the breakpoint column



Breakpoints – Maintain Breakpoints

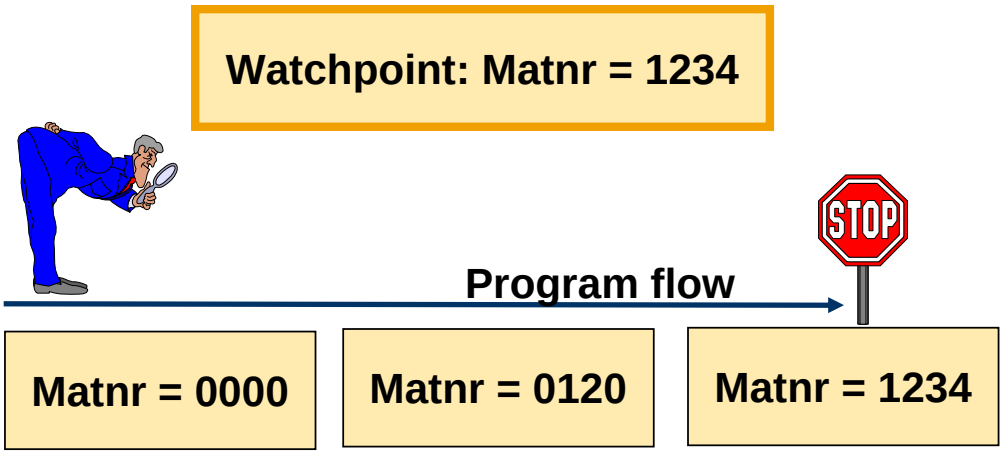


- In the break-point maintenance view you can easily delete/create/activate/deactivate your breakpoints.
- Additionally you may convert your breakpoints from debugger to session breakpoints etc.
- Double-clicking on the navigation button displays the breakpoint in the source code.

Watchpoints

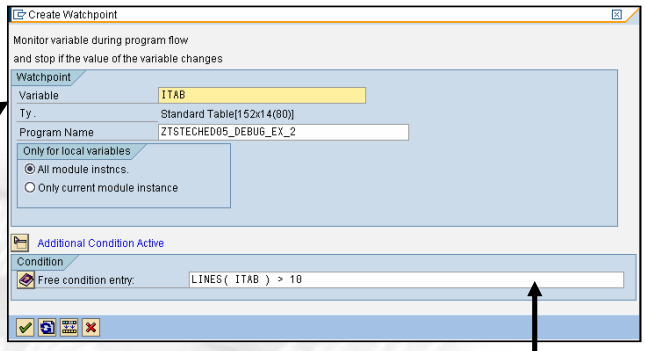
If you set a watchpoint on the field matnr (matnr = '1234'), the debugger stops as soon as matnr contains the value '1234'.

If you do not specify a condition, the debugger stops as soon as the field matnr changes.



- Watchpoints are very dynamic breakpoints.
  You can specify a field that should be monitored and you can specify a stop condition.
- The watchpoints of the New ABAP Debugger are not shared with the Classic Debugger. This is a totally new implementation which assures correct behavior of the
  watchpoints for all kind of variables (including object attributes, local variables and even internal tables)

Create a Watchpoint



- **specify variable**
  - **stop on change of variable**
  - **all global, local variables**
  - **including internal tables**
  - **not possible: debugger symbols pointing into internal tables**
    e.g.: “itab[1]-comp”

- **specify variable & condition**
  - **stop when variable changes AND condition is true**
  - **condition restricted to two operands and 1 operator**
  - **built-in functions (“lines( )” and “strlen( )”) possible**
    e.g.: “sy-subrc <> 0” or “strlen(string) > 10”

- **can be activated / inactivated**

THE BEST-RUN BUSINESSES RUN SAP™

Classic Debugger: No watchpoints on internal tables allowed, but…

- You can use a watchpoint on the table header of an internal table to watch at least the line count of an internal table:
- Each internal table has a header which contains administration data like number of lines.
  If, e.g., someone deletes lines or appends lines, then this header is updated.
  It is possible to set a watchpoint on the table header. This is very helpful if you want, e.g., find the source line where an internal-table line is deleted.
  But keep in mind that modifications of the contents in a line have normally no effect on the table header.
  There is some more info stored in the table header (like the loop counter); therefore don't be too astonished if the program stops, e.g., at a LOOP statement.
- Watchpoints on table headers in different kernel releases:

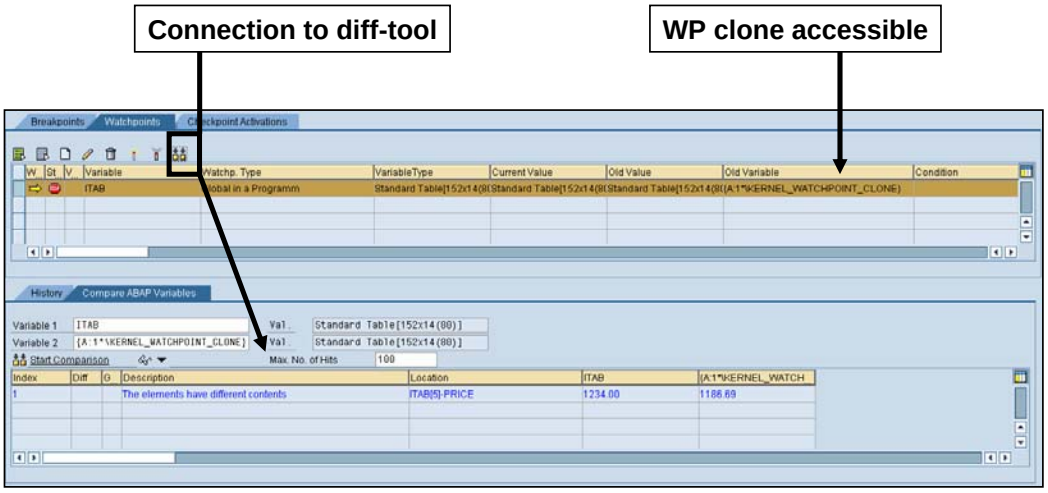
< 40B	itab-*sys*
40B	itab[]
45B	itab[]+0(128)
since 46D	*itab[] (header is only filled when table is not initial!)

Since kernel release 46D the table header is no longer stored directly in the addressable memory. Only a table reference (8bytes) remains in the memory addressable by the user. This table reference is a pointer which points to the header

Watchpoints

When stopping at a watchpoint

- **value of “old” variable is available (reference to WP clone)**
- **Diff-tool can be used to be guided to difference(s)**



- In contrast to the Classic Debugger the New ABAP Debugger creates always a clone of the watch-point variable .
- After reaching a watchpoint the clone contains the status of the old variable and the current variable contains the current status.
- Using this clone technique, you can now compare these two variables –e.g. by using the diff tool to to find out where the variable was changed.

Copyright 2005 SAP AG. All Rights Reserved