



# ABAP Debugging – 25 Real-Time Challenges & Solutions

#	Challenge (Real-time Scenario)	Solution (Detailed Explanation)
1	Debugging standard SAP code without modifying it.	Use <b>external breakpoints</b> (Shift+F9) for background or Fiori users. Set breakpoints in <b>implicit enhancement points</b> or <b>BAdI implementations</b> . If debugging standard logic, enable <b>System → Utilities → Breakpoints → External</b> and log in as the same user in another session.
2	Debugging background jobs (SM37) that fail without short dump.	Run the same program via <b>SA38 → Execute in Foreground</b> if possible. Else, go to <b>SM37 → Job → Set Debugging</b> before start. Alternatively, use <b>SM50 → Program/Mode → Program/Session → Debugging</b> to attach debugger mid-execution.
3	Debugging update tasks (CALL FUNCTION ... IN UPDATE TASK).	Set breakpoint in the <b>update function module</b> and use <b>Update Debugging</b> in menu: <i>System → Utilities → Debugging → Update Debugging ON</i> . Execute the main transaction and debugger will stop when the update task triggers.
4	Debugging workflow-triggered function modules or methods.	In <b>SWI1</b> , note the task's function module. Set external breakpoint in that module or method. Then, in workflow runtime, choose <i>Restart with Debugging</i> or use <b>SWUD → Execute with Debugging</b> .
5	Debugging Fiori/OData calls from the Gateway.	Go to <b>/IWFND/GW_CLIENT</b> , execute the OData call, and debug backend logic by setting <b>external breakpoints</b> in <b>DPC_EXT</b> or <b>RAP behavior class</b> . Ensure the breakpoint user matches the Fiori user.
6	Debugging RFC/BAPI calls made from another system.	Use <b>SM59 → Utilities → Test → Connection</b> to identify RFC destination. Set <b>external breakpoint</b> in the RFC-enabled FM and call it from the external system. Ensure <b>user and client</b> are same in both systems.
7	Debugging background Fiori OData batch (\$batch) requests.	Place <b>dynamic breakpoints</b> in each <b>DPC_EXT</b> method. Use <b>/IWFND/TRACES</b> to identify the internal method sequence, then reproduce request in <b>/IWFND/GW_CLIENT</b> . Activate <b>External Debugger</b> for the same user.
8	Debugging inbound IDocs that fail in background.	Use <b>WE02</b> to find the IDoc number. Execute <b>BD87</b> , select the failed IDoc, and click <i>Process → Foreground</i> . When the function module (e.g., <b>IDOC_INPUT_XXX</b> ) triggers, debugger will stop at your breakpoints.
9	Debugging outbound IDocs generated automatically.	Use <b>WE05 → Outbound → Display</b> and note the message type. Set breakpoint in <b>MASTER_IDOC_DISTRIBUTE</b> . Trigger event again. For

#	Challenge (Real-time Scenario)	Solution (Detailed Explanation)
10	Debugging BADI implementations triggered dynamically.	partner profile calls, debug NACE → <b>Processing Routine</b> assigned to message. Use <b>CL_EXITHANDLER</b> → method <code>GET_INSTANCE</code> . Set breakpoint there to see which BAdI is being called. You can then double-click the implementation class to continue debugging inside it.
11	Debugging dynamic function module or class method calls.	When you see <code>CALL FUNCTION (lv_func) OR CALL METHOD (lv_class)=&gt; (lv_method)</code> , set a <b>breakpoint at statement</b> (Shift+F9 → "Break/Watchpoint → Breakpoint at → Statement → CALL FUNCTION/METHOD"). Then inspect variable names at runtime.
12	Debugging standard BAPI commits (e.g., <code>BAPI_PO_CREATE1</code> not saving).	Standard BAPIs need explicit <code>BAPI_TRANSACTION_COMMIT</code> . Set breakpoint there and check if <code>WAIT</code> parameter is passed. If not, commit might be skipped in batch. Use <b>Update Debugging</b> to follow up.
13	Debugging issues that occur only in user-specific variants or authorizations.	Run the transaction as the <b>affected user</b> using <code>/oUsername</code> in SU53 or SU56, and set <b>external breakpoints</b> . Use <b>Authorization Trace (ST01)</b> alongside debugger to confirm missing roles.
14	Debugging batch input (BDC) sessions.	Go to <b>SM35</b> , select the session, choose <i>Process</i> → <i>Foreground</i> . The debugger can be started by setting <code>/h</code> before pressing <i>Enter</i> on any screen. Then step through each screen field mapping. For Smartforms, set breakpoint in the <b>generated function module</b> (find via SMARTFORMS → Environment → Function Module Name). For Adobe Forms, set breakpoint in <b>interface FM or form processing class</b> .
15	Debugging Smartforms or Adobe Forms.	Set <b>update debugging + system debugging</b> ON. Then, in SM50, find the new task process → <i>Administration</i> → <i>Program/Mode</i> → <i>Debugging</i> . Attach to that session dynamically.
16	Debugging in parallel processing ( <code>CALL FUNCTION ... STARTING NEW TASK</code> ).	Set breakpoint in <code>ENQUEUE_E / DEQUEUE_E** FM</code> . Step through lock object parameters. Also use <b>SM12</b> to check current locks. In race conditions, simulate two sessions updating the same record.
17	Debugging enqueue/dequeue (locking) issues.	Set breakpoint in <b>DBSQL_SQL_EXECUTE</b> or <b>CL_SQL_STATEMENT-&gt;EXECUTE_QUERY</b> . Check SQL text via debugger variable <code>stmt-&gt;sql_text</code> . Compare to expected SELECT to detect SQL injection or runtime syntax issues.
18	Debugging dynamic SQL execution (native SQL or EXEC SQL).	Use <b>watchpoints</b> on <code>&lt;fs&gt;</code> and <code>ASSIGN</code> statements. In debugger, use "Dereference" (Ctrl+Shift+F7) to see target variable contents. For data refs, use <code>(-&gt;*)</code> to inspect.
19	Debugging field-symbols & data references (runtime assignments).	

#	Challenge (Real-time Scenario)	Solution (Detailed Explanation)
20	Debugging performance-heavy loops without slowing system.	Use <b>Debugger Scripting</b> (Transaction SDBG → Scripts Tab). Write small scripts to log variable values automatically at specific lines instead of stepping manually. Or use <b>SAT trace</b> combined with conditional breakpoints.
21	Debugging custom enhancement points inside standard code.	Activate enhancement in debug mode (Shift+F9 → Breakpoint in include). If enhancement doesn't trigger, ensure it's <b>active</b> (check via SE80 → Enhancement Implementation → Activate).
22	Debugging RAP (Managed/Unmanaged) logic.	Set breakpoints in <b>Behavior Pool Class (BP_I_XXXX)</b> methods. For managed, use <code>before save</code> , <code>after modify</code> , or <code>determination</code> methods. Trigger via Fiori preview or EML. In Eclipse, use “Start Debugger for HTTP Requests”.
23	Debugging errors during transport or activation (syntax check bypass).	In SE10 → transport logs, identify object. Open object → activate manually in debug mode. Use SE80 → Utilities → Syntax Check → Activate Debugger. Also use SPDD or SPAU if dictionary objects are involved.
24	Debugging memory leaks or unassigned references.	Use <b>Memory Inspector (S_MEMORY_INSPECTOR)</b> or <b>SAT → Memory Allocation Trace</b> . Compare snapshots before and after execution. Use debugger variable <code>GET_REFERENCE</code> to see where memory remains allocated.
25	Debugging cases that crash without dumps (silent termination).	Activate <b>system debugging</b> and <b>Update Debugging</b> , then enable <b>ST05 SQL trace + SMICM HTTP trace</b> for deeper investigation. Sometimes, <b>CL_CATCH_ALL exceptions</b> consume dumps silently — set breakpoint in <code>CX_ROOT-&gt;RAISE_EXCEPTION</code> to trap it.



## Debugging SME Golden Practices

1. **Always start small.** Narrow down issue scope using short test cases (unit tests, single data records).
2. **Activate all layers of debugging** when unsure: Update, System, HTTP, and External.
3. **Use Watchpoints** on key data changes instead of stepping through every line.
4. **Combine Debugger + ST05 + SAT + ST12** — these four are your *investigation toolkit*.
5. **Debugger scripting** (since NW 7.52) can automate recurring trace collection tasks.
6. **Dynamic breakpoints in CALL FUNCTION/CALL METHOD** uncover hidden runtime polymorphism.
7. **For BTP or RAP debugging**, use Eclipse ADT debugger with HTTP session mapping.
8. Always document complex debugging findings in a central wiki — they become internal assets.