



ABAP Debugging – 25 Real-Time Challenges & Solutions

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

#	Challenge (Real-time Scenario)	Solution (Detailed Explanation)
		partner profile calls, debug NACE → Processing Routine assigned to message.
10	Debugging BADI implementations triggered dynamically.	Use CL_EXITHANDLER → 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)</code> or <code>CALL METHOD (lv_class)=> (lv_method)</code> , set a breakpoint at statement (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 Update Debugging to follow up.
13	Debugging issues that occur only in user-specific variants or authorizations.	Run the transaction as the affected user using <code>/oUsername</code> in SU53 or SU56, and set external breakpoints . Use Authorization Trace (ST01) alongside debugger to confirm missing roles.
14	Debugging batch input (BDC) sessions.	Go to SM35 , 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.
15	Debugging Smartforms or Adobe Forms.	For Smartforms, set breakpoint in the generated function module (find via SMARTFORMS → Environment → Function Module Name). For Adobe Forms, set breakpoint in interface FM or form processing class .
16	Debugging in parallel processing (<code>CALL FUNCTION ... STARTING NEW TASK</code>).	Set update debugging + system debugging ON. Then, in SM50, find the new task process → <i>Administration</i> → <i>Program/Mode</i> → <i>Debugging</i> . Attach to that session dynamically.
17	Debugging enqueue/dequeue (locking) issues.	Set breakpoint in <code>ENQUEUE_E / DEQUEUE_E** FM</code> . Step through lock object parameters. Also use SM12 to check current locks. In race conditions, simulate two sessions updating the same record.
18	Debugging dynamic SQL execution (native SQL or EXEC SQL).	Set breakpoint in <code>DBSQL_SQL_EXECUTE</code> or <code>CL_SQL_STATEMENT->EXECUTE_QUERY</code> . Check SQL text via debugger variable <code>stmt->sql_text</code> . Compare to expected SELECT to detect SQL injection or runtime syntax issues.
19	Debugging field-symbols & data references (runtime assignments).	Use watchpoints on <code><fs></code> and <code>ASSIGN</code> statements. In debugger, use "Dereference" (Ctrl+Shift+F7) to see target variable contents. For data refs, use <code>(->*)</code> to inspect.

#	Challenge (Real-time Scenario)	Solution (Detailed Explanation)
20	Debugging performance-heavy loops without slowing system.	Use Debugger Scripting (Transaction SDBG → Scripts Tab). Write small scripts to log variable values automatically at specific lines instead of stepping manually. Or use SAT trace 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 active (check via SE80 → Enhancement Implementation → Activate).
22	Debugging RAP (Managed/Unmanaged) logic.	Set breakpoints in Behavior Pool Class (BP_I_XXXX) 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 Memory Inspector (S_MEMORY_INSPECTOR) or SAT → Memory Allocation Trace . 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 system debugging and Update Debugging , then enable ST05 SQL trace + SMICM HTTP trace for deeper investigation. Sometimes, CL_CATCH_ALL exceptions consume dumps silently — set breakpoint in <code>CX_ROOT->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.