The New
ABAP Debugger
- An Introduction

Boris Gebhardt
Christoph Stoeck
SAP AG
Content

- Motivation & Goals
- Two Process Architecture
- Starting The New Debugger
- New Debugger UI – Main Parts
- Customize The New Debugger UI
- Debugger Tools
- Breakpoints
- Error Handling
- Open points & outlook
- Exercise
- Q&A
Motivation

We have already a powerful ABAP debugger.

Why do we need a new one??

We have already a powerful ABAP debugger.

Why do we need a new one??
Demo

- Conversion exit demo TPDA_CONV_EXIT2 with classic debugger
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 !)

We need a new ABAP debugger technology
New ABAP Debugger – First Impression

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

233 234 235 236 237 238
239 CREATE OBJECT t_ref.
240 REFRESH.

Table T84B

<table>
<thead>
<tr>
<th>MarketID</th>
<th>Carried</th>
<th>Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>200</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>300</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>400</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>500</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>600</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>700</td>
<td>A</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table T841

<table>
<thead>
<tr>
<th>MarketID</th>
<th>Carried</th>
<th>Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>100</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>200</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>300</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>400</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>500</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>600</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td>700</td>
<td>A</td>
<td>Y</td>
</tr>
</tbody>
</table>
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
The New Debugger is attached to a session

Session 1 - Debuggee

Session 2 - Debugger
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
• 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!
Demo

- Conversion exit demo TPDA_CONV_EXIT2 with new debugger
Motivation & Goals
Two Process Architecture
Starting The New Debugger
New Debugger UI – Main Parts
Customize The New Debugger UI
Debugger Tools
Breakpoints
Error Handling
Open Points & Outlook
Exercise
Q&A
Starting The New Debugger

The New Debugger can be started with same commands (/h, “Debugging” in se38 ...) as the Classic Debugger.

- There is a system parameter to define the default debugger for the whole system. (Default = Classic Debugger)
  
  abap/DebuggerTool = TPDA (->New Debugger)

- Additionally each user can specify his default debugger in the workbench settings.
Switching Between New <-> Classic Debugger

Some techniques and features are still missing in the New Debugger
-> Easy switch between the two debugger variants is provided

- RPERF_ILLEGAL_STATEMENT Dump will come up, if you switch to the classic debugger with a conversion- or Field exit
New Debugger UI – Main Parts

- Control Area
- Process Info
- Source line / SY-Fields
- Desktops
- Tools
New Debugger UI – Desktops

User specific desktops

| Desktop 1 | Desktop 2 | Desktop 3 |

Standard desktops

| Standard | Structures | Tables | Objects | Detail Displs | Breakpoints |

The New Debugger provides:

- Three user specific desktops, which you can customize and save as your favorite debugger environment
- Six standard desktops, which should cover most of the common working conditions in the debugger:
  - Standard: Stepping through the code (Editor, Stack, Quick Watch)
  - Structures: Compare structures
  - Tables: Compare tables
  - Objects: Compare objects
  - Detail Displs: Compare strings, simple fields ...
  - Breakpoints: Maintain your breakpoints

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

- Demo: demonstrate desktops -> compare two internal tables (2bgtest_itab/itab/itab2)
Motivation & Goals
Two Process Architecture
Starting The New Debugger
New Debugger UI – Main Parts
Customize The New Debugger UI
Debugger Tools
Breakpoints
Error Handling
Open Points & Outlook
Exercise
Q&A
Customize The New Debugger UI – Toolbar

With the normal “Back” button (F3) you can “Undo” all your layout changes

Context menu

Close Tool
Create Tool
Exchange Tool
Full screen mode
Maximize horizontally
Swap Tool
Services of the tool

ABAP Source Code Display

<table>
<thead>
<tr>
<th>Br.</th>
<th>P.</th>
<th>Row</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>223</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td></td>
<td>224</td>
<td>* -- p1 text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>225</td>
<td>* &lt;- p2 text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>226</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>227</td>
<td>FORM abc .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>228</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>229</td>
<td>DATA: BEGIN OF struct,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>230</td>
<td>a TYPE c,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>231</td>
<td>i TYPE i,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>232</td>
<td>END</td>
</tr>
</tbody>
</table>

© SAP 2004
- Debug a standard transaction (se80) and demonstrate the UI customizing possibilities
Motivation & Goals
Two Process Architecture
Starting The New Debugger
New Debugger UI – Main Parts
Customize The New Debugger UI
Debugger Tools
Breakpoints
Error Handling
Open Points & Outlook
Exercise
Q&A
Debugger Tools

• Up to 4 parallel visible instances of a tool (e.g. compare 4 internal tables)

• Tools are integrated in debugger framework and benefit from the provided services to customize the UI. (Change size, position, close tool, …)

• Tools provide standard services
If a tool provides services then a tool icon is displayed in the toolbar.

The standard services are
- Download of the tool content to a local file
- Search in specified columns (STRG+F)

Each tool can add its own additional services
Debugger Tools – Standard Tools

Standard tools:
- Source display
- Stack
- Breakpoints
- Variables

- Detail views
  - Structure
  - Internal table
  - Object
  - Simple types
Navigation between ABAP data structures

- Double-click in the Editor fills appropriate Detail View and fills/opens Quick Watch
- Double-click in Quick Watch fills/opens appropriate Detail View
- Double-click in Detail View exchanges current tool by appropriate Detail View
Debugger Tools – New Tools - Architecture

Debugger Engine

ADI

Debugger UI framework

TOOL_FACTORY
OK_CODE_HANDLER
SUBSCREEN_HANDLER

Tool interface

Stack  ...  Editor  Breakpoints

Tools

Tool class
IF_TPDA_TOOL

Function group - screens
Extensibility (even plug in of new tools)

- Create the class and the necessary function groups
- Implement interface IF_TPDA_TOOL and if necessary IF_TPDA_TOOL_DETAIL_VIEW
- Copy the data transfer mechanism between instances and dynpro from existing tools
- Implement your functionality in the tool class and design the screens
- Register your tool for other events you are interested in
- Insert a new entry into New Debugger customizing tables, to register your tool
Demo

- Debug zbgtest_itab ->
  - Services (search, download ...)
  - Navigation
  - Features of editor, stack
  - OO View
  - Detail view (translation)
  - Change variables (structure, table...)
Breakpoints

- Classic Debugger breakpoints live in an internal sessions
  These breakpoints are no longer available, if a new internal session is created (SUBMIT, CALL TRANSACTION, ...)

- New Debugger breakpoints live in an external sessions
  These breakpoints are lost when the debuggee session is closed.
- Debug RSDEPEND (breakpoint at write) with classic debugger
Create/Deactivate/Delete a breakpoint in the Editor by double-clicking on a line.

Create dynamic breakpoints.
Breakpoints – Maintain Breakpoints

Create/Delete/Activate/Deactivate breakpoints

Specify how often the breakpoint shall be skipped before stopping
- Debug RSDEPEND with new debugger (break-point at write)
- (set breakpoint at FORM=WRITE_DESCR / PRG=RSPFPAR)
Content

Motivation & Goals
Two Process Architecture
Starting The New Debugger
New Debugger UI – Main Parts
Customize The New Debugger UI
Debugger Tools
Breakpoints
Error Handling
Open Points & Outlook
Exercise
Q&A
Error Handling

The ABAP debugger is a very important tool which must run ALWAYS.

Fall back in case of error:

- An error during the initialization of New Debugger (e.g. no tools found) occurs.
  The user can switch to Classic Debugger.

- New Debugger catches all exceptions raised by the tools. In case of a severe error, the tool will be closed.

- An error, or even an application dump in the New Debugger UI itself, does not influence the debuggee.
  The UI of New Debugger is restarted from scratch and the user can continue debugging.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two session debugger technology</td>
<td>Already implemented</td>
</tr>
<tr>
<td>Standard debug features (breakpoints, variables, stack,…)</td>
<td>Already implemented</td>
</tr>
<tr>
<td>UI – framework and sophisticated standard tools</td>
<td>Already implemented</td>
</tr>
<tr>
<td>Watchpoints</td>
<td>Not yet implemented</td>
</tr>
<tr>
<td>Debugging of special sessions: batch, RFC, update, HTTP</td>
<td>Not yet implemented</td>
</tr>
<tr>
<td>Special features of Classic Debugger (system areas, …)</td>
<td>Not yet implemented</td>
</tr>
<tr>
<td>New Debugger becomes default</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>➢ Match functionality of Classic Debugger</td>
<td></td>
</tr>
<tr>
<td>➢ Further enhancements</td>
<td></td>
</tr>
<tr>
<td>➢ Debugger Engine</td>
<td></td>
</tr>
<tr>
<td>➢ Sophisticated dynamic breakpoints</td>
<td></td>
</tr>
<tr>
<td>(breakpoint at special exception, event, message, program)</td>
<td></td>
</tr>
<tr>
<td>➢ Debug macros</td>
<td></td>
</tr>
<tr>
<td>➢ New debugger UI</td>
<td></td>
</tr>
<tr>
<td>➢ Trace Tool (SQL Trace, SE30, ST01)</td>
<td></td>
</tr>
<tr>
<td>➢ Dynpro Tool (SCREEN, attributes of screen elements)</td>
<td></td>
</tr>
<tr>
<td>➢ Diff tool for all detail views</td>
<td></td>
</tr>
<tr>
<td>➢ …</td>
<td></td>
</tr>
</tbody>
</table>
Content

- Motivation & Goals
- Two Process Architecture
- Starting The New Debugger
- New Debugger UI – Main Parts
- Customize The New Debugger UI
- Debugger Tools
- Breakpoints
- Error Handling
- Open Points & Outlook
- Exercise
- Q&A
Exercise I

System / logon info:
SYSTEM W01 (sapwas01.wdf.sap.corp/00)
CLIENT 100
USER DEMOxx (01-10)
PASSWORD TRAIN (or TRAINING change it to TRAIN please)

Preparation:
- Change your workbench user settings to “New Debugger”
- Start debugging report ZBGTEST_S3_EX within SE38.

The exercises in report ZBGTEST_S3_EX start all with a hard coded breakpoint so that you can reach them easily in the debugger.

Exercise 1: Layout customizing
(a) Play with the layout customizing of the New Debugger
e.g. try to maximize the Editor horizontally in order to see the full line length
-> switch to the line length 72 using the service menu of the editor
(b) Save your layout and restart the debugger to see that your new settings are loaded automatically
Exercise II

Exercise 2: displaying / changing variables

(a) Change the variables of the IF clause:
   If struc = 'AA' and sy-subrc = 9 and sy-title is initial.
   so that you see the message on the debuggee screen.

(b) Try to use „Jump to statement“ to get directly to the message without the need of changing variables.

(c) Try to jump back to the message statement after receiving the message

Exercise 3: Object Detail View

(a) When we create an instance of CL_GUI_ALV_GRID, we transfer the object reference CONTAINER.
   create object alv_control exporting i_parent = container.
   Please find out in which attribute of ALV_CONTROL the object reference CONTAINER is stored.
   Hint: Display the object reference CONTAINER and check the “Display Reference Tab”

(b) Display ALV_CONTROL (inheritance) to find out why one of the following casts did not work.
   ref2 ?= alv_control.
   ref ?= alv_control.
Exercise 4: Table View / Breakpoints

(a) Display ITAB in the table view (full screen), when you reached the first breakpoint of exc. 4

- How many lines are in the table – check the content of the different columns

(b) This internal table ITAB, which contains the content of DB table SFLIGHT, is exported to the ABAP memory and afterwards we submit a second report

- Continue until you reach the second hard coded break-point of exc. 4 and check the table ITAB which is imported from our ABAP memory.

- Try to find out how and why the table ITAB differs from our original SFLIGHT table (ITAB2) which we stored in the ABAP memory
Exercise IV

Exercise 5: Table View / Breakpoints - Hint

Set a breakpoint after the first export to memory.

Run to this point and set a dynamic breakpoint at EXPORT in order to find potential overwriters of our ABAP memory.

After finding the EXPORT in function TPDA_MY_CONVERSION2, which overwrites our ABAP Memory with another table ITAB(1. row is different), correct the internal table ITAB before the EXPORT. (row 1 / MANDT = XX ->100)

You will see that after this correction we reach the branch ITAB = ITAB2 and everything is ok.
Optional Exercise 5: Detail View for single fields

The content of a file is loaded in two long hex fields.
Both hex-fields are always manipulated identically and in parallel, but when we compare them at the end, they differ.
Try to find out where the difference comes from.

Hint:
The program writes 123456789 in the text which is represented by the hex fields.
Use the detail view for single fields and here the “tabular” and “translation” view for both fields in parallel and try to find the difference.
(Use code page = 1100 for the “translation view”)
Take care of the variables X and X2 in report ZBGTEST_S3_EX2…
Content

- Motivation & Goals
- Two Process Architecture
- Starting The New Debugger
- New Debugger UI – Main Parts
- Customize The New Debugger UI
- Debugger Tools
- Breakpoints
- Error Handling
- Open Points & Outlook
- Exercise
- Q&A
Questions?

Q&A