ABAP Debugging Tips and Tricks

Applies to:

This article applies to all SAP ABAP based products; however the examples and screen shots are derived from ECC 6.0 system. For more information, visit the ABAP homepage.

Summary

This article gives a list of useful tips and tricks which will make debugging and coding easier.

Author: Sai Santosh K

Company: IBM India Pvt. Ltd.

Created on: 6th July 2009

Author Bio

Sai Santosh is a Sr. Consultant working in IBM India Pvt. Ltd.; he has 6 years of SAP programming experience in various SAP solutions including SD, CRM Channel Management and HR.

Table of Contents

Debugging Tips and Tricks

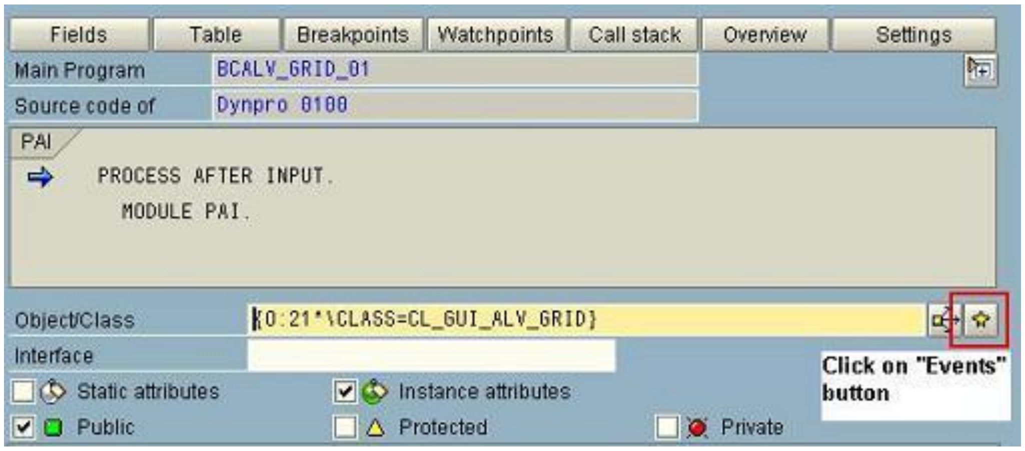
SAP provides a very powerful debugger, knowing all its features would make debugging fun and fast. Below I list out a few relatively hidden features of the debugger which I have personally found to be very helpful. I discovered these features either by exploration, came across them on the web or via colleagues.

Finding event handlers

Class methods can be registered as event handlers dynamically at run time, we might be interested in finding which method is registered as the event handler for an event, and SAP debugger provides a way to determine this.

Using Classic debugger

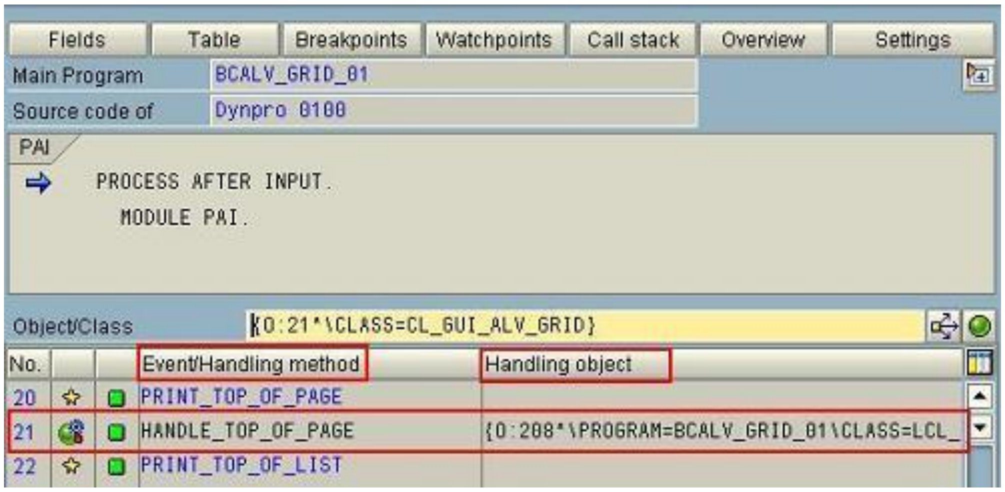
Double click on the variable to reach the Object/Class display section, now click on the "*Events*" button, this displays a list of events raised by the class and their corresponding handlers.



Main Program: BCALV_GRID_01
Source code of: Dynpro 0100

```
PAI
    PROCESS AFTER INPUT.
      MODULE PAI.
```

Object/Class: {O:21*\CLASS=CL_GUI_ALV_GRID}
Interface:



Main Program: BCALV_GRID_01
Source code of: Dynpro 0100

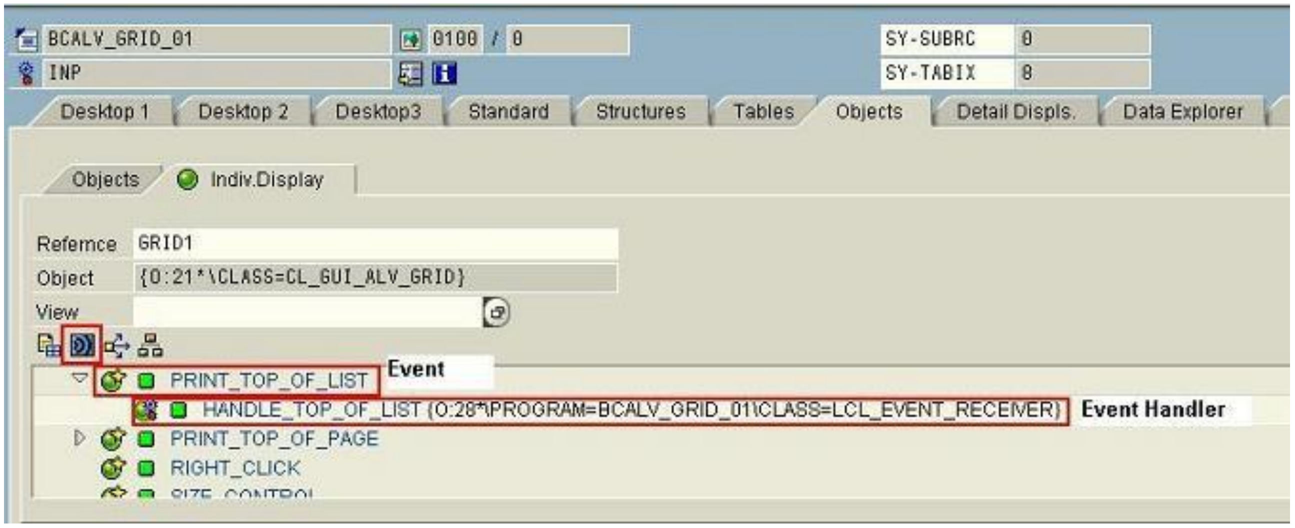
```
PAI
    PROCESS AFTER INPUT.
      MODULE PAI.
```

Object/Class: {O:21*\CLASS=CL_GUI_ALV_GRID}

No.			Event/Handling method	Handling object
20	☆	🟩	PRINT_TOP_OF_PAGE	
21		🟩	HANDLE_TOP_OF_PAGE	{O:208*\PROGRAM=BCALV_GRID_01\CLASS=LCL_
22	☆	🟩	PRINT_TOP_OF_LIST	

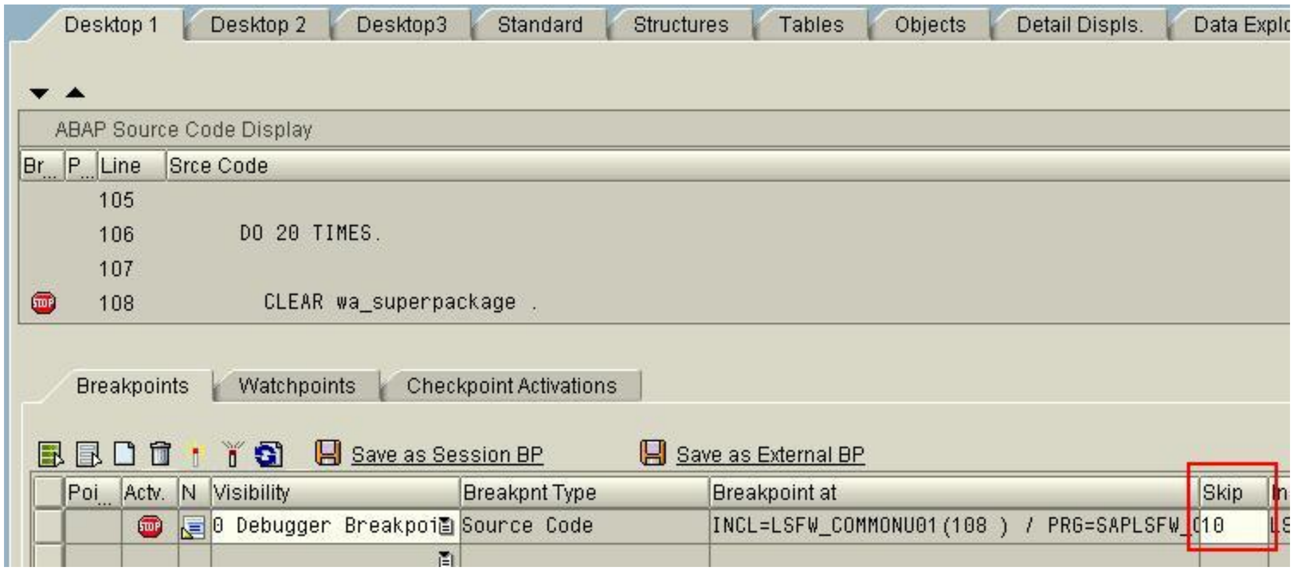
Using New debugger

Key in the reference variable in the “*Objects*” tab, now click on the “*Display Events” button* this displays a list of events raised by the class and on expanding the dropdown their corresponding handlers are shown.



Skip breakpoints

The new ABAP debugger gives an option to skip the first *n* occurrences of a breakpoint; this would be useful while debugging loops as we might want to skip the first *n* executions of the loop. Or the first *n* times a function module / subroutine is called. The below screen shot shows where we can enter the number of times the breakpoint should be skipped.



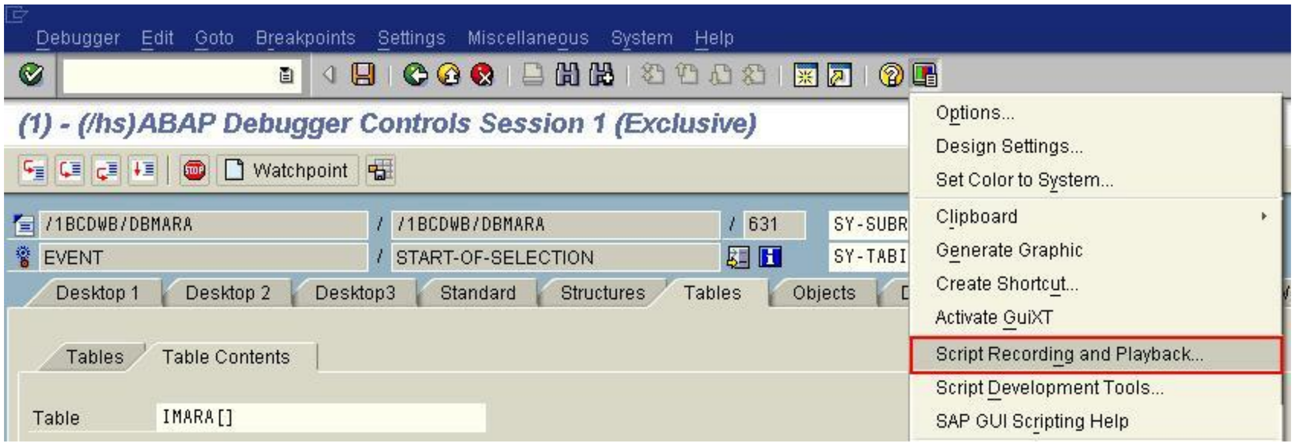
```
Br. P. Line  Srce Code
       105
       106        DO 20 TIMES.
       107
       108          CLEAR wa_superpackage .
```

Poi..	Actv.	N	Visibility	Breakpnt Type	Breakpoint at	Skip
			0 Debugger Breakpoi	Source Code	INCL=LSFW_COMMONU01(108) / PRG=SAPLSFW_C	10

Note: The breakpoint tool can be displayed either by adding it as new tool or navigating to the “*Break/Watchpoints*” tab.

Script recording and playback

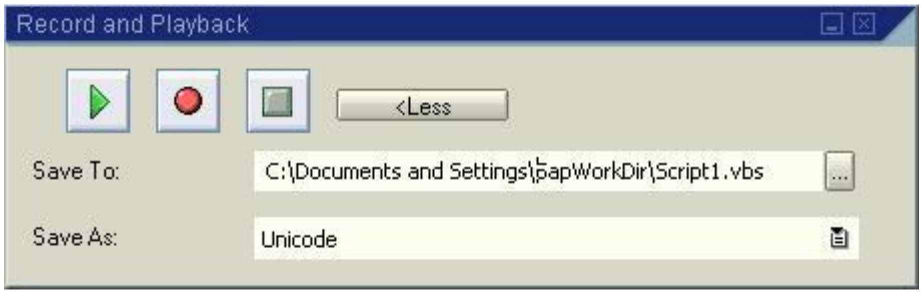
While debugging complex programs, or testing a user exit, we generally have to perform a set of repetitive actions in the debugger before reaching the point in code we are actually interested in. The actions might be anything from launching a transaction to adding a few lines to an internal table. Any action we perform on the SAP GUI can be automated using Script Recording & Playback!! Just start the "Script recorder" by pressing the "*Customize Local Layout"* button (Alt + F12) and select "*Script Recording and Playback…*" as shown below. Then hit the "*Record Script*" button and perform the set of actions as you would generally do them. The VB script gets saved in the default SAP work directory. You can play it back using the "*Playback script*" button and the entire set of operations is performed automatically!!



/1BCDWB/DBMARA / /1BCDWB/DBMARA / 631 SY-SUBR

EVENT / START-OF-SELECTION SY-TABI

Table IMARA[]



Save To: C:\Documents and Settings\SapWorkDir\Script1.vbs

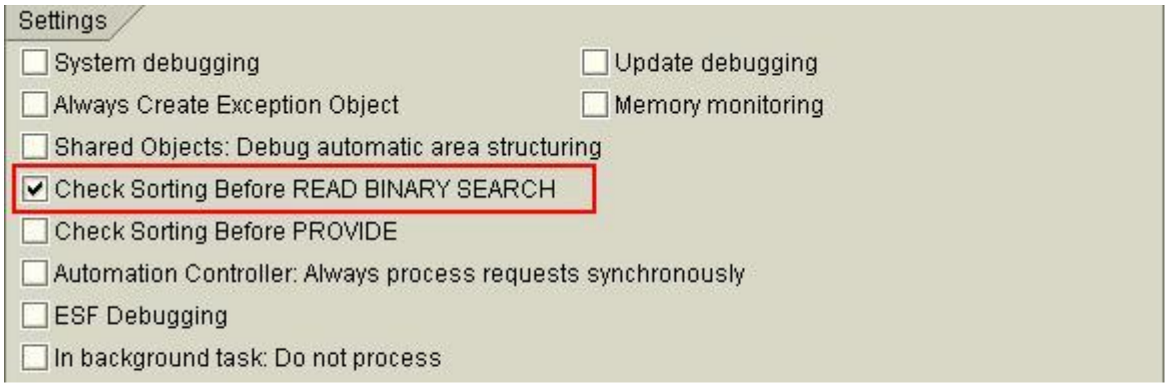
Save As: Unicode

Note: Scripting has to be enabled on your SAP system for this to work. Enabling scripting on a production system is **NOT** recommended.

Check sorting before READ BINARY SEARCH

Before delivering the code, we can use this option to make sure all internal tables are properly sorted before READ BINARY SEARCH is performed on them. If the table is not sorted a runtime error is generated. This ensures that the READ statement does not return incorrect results.

Path (old debugger): Settings tab



Settings

- [ ] System debugging
- [ ] Update debugging
- [ ] Always Create Exception Object
- [ ] Memory monitoring
- [ ] Shared Objects: Debug automatic area structuring
- [✓] Check Sorting Before READ BINARY SEARCH
- [ ] Check Sorting Before PROVIDE
- [ ] Automation Controller: Always process requests synchronously
- [ ] ESF Debugging
- [ ] In background task: Do not process

Debug RFC calls

When an RFC function module is called it is usually not possible to debug the call, using the below techniques we can achieve the same. For this example we are calling an RFC enabled FM in ECC system from CRM system.

Scenario 1: Calling a custom RFC FM

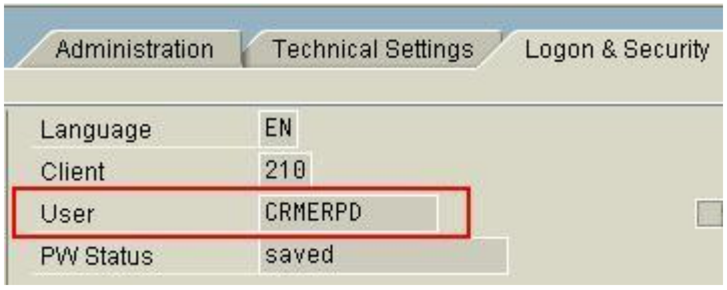
STEP 1: Add an infinite loop in the remote custom Function Module (adding a “DO. ENDDO.” statement at the start of the FM would be enough).

STEP 2: Execute your program in the CRM system. The execution stops at the RFC call, because of the infinite loop.

STEP 3: Now login to the ECC system and go to transaction SM51 select the process which is executing the RFC and navigate to the menu: *“Program/Session->Program->Debugging”* this triggers the debugger session in a separate window.

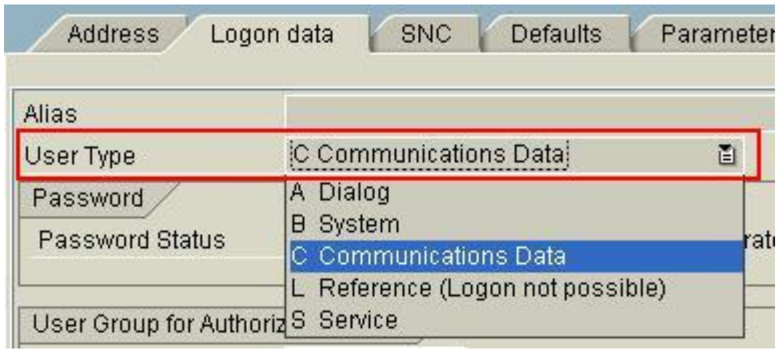
Scenario 2: Calling a Standard RFC FM

When we are debugging a standard program we cannot add the infinite loop, hence we cannot go to SM51 and debug the work process. The RFC destination for the ECC system has a User specified, the RFC function module gets executed using this users credentials. Usually the basis team set up the RFC user as non Dialog user, this does not allow debugging. Contact your Basis team and get this changed to a dialog user enabling you to debug RFC function calls via the normal debugger.



Language	EN
Client	210
User	CRMERPD
PW Status	saved

Note: User name configured in the RFC destination (transaction SM59)



User Type: C Communications Data

Note: User type for the RFC User (transaction SU01D)

Debugging a Popup Window

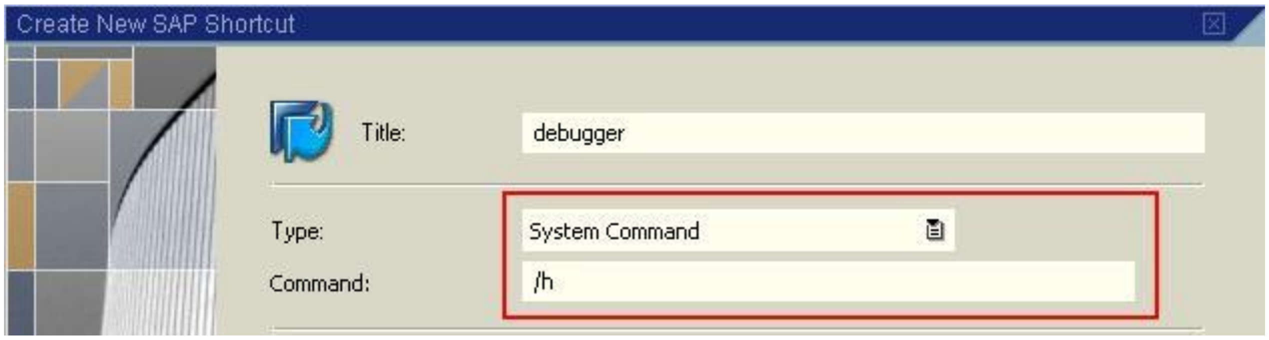
Sometimes we might want to start debugging from a popup window / information message, in this case we cannot type '/h' as the "OK Code" box is not available.

Method 1: For this create a text file (say debugger.txt) on your desktop, and type the below lines into the file:

```
[FUNCTION]
Command=/H
Title=Debugger
Type=SystemCommand
```

Now drag the file onto the popup window / information message and debugging is enabled!

Method 2: From any SAP windows press the "*Customize Local Layout"* button (Alt + F12) and select "*Create Shortcut…*" in the window that appears make the below changes and click "*Finish".* A file is generated on your desktop or SAP workspace. Now drag the file onto the popup window / information message and debugging is enabled!



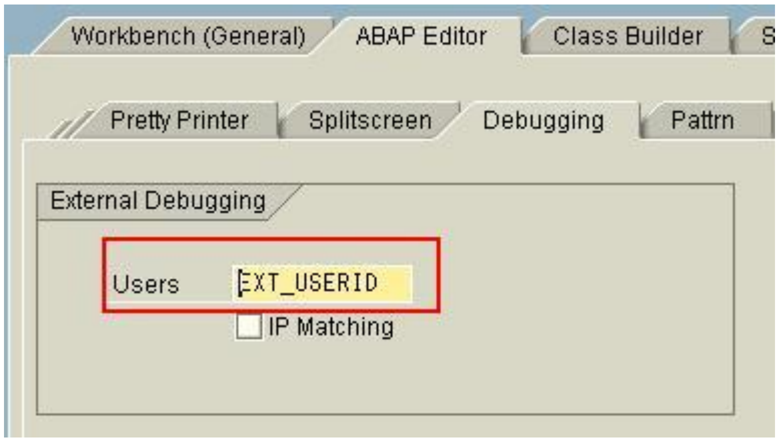
Title: debugger

Type: System Command

Command: /h

External breakpoints

In order to debug ABAP code which is called via Portal or any external system you can need to follow a two step approach. Firstly enter the user ID using which you would login to the portal / HTTP application, this can be done in on of the workbench development utilities (SE38, SE37, SE80, etc) choose *Utilities* → *Settings* → *Debugging* → *External Debugging*. Secondly set external breakpoint at the required position in the ABAP code using the *Set/Delete External Breakpoints* button. Now when the external application is run the ABAP debugger opens in a separate window. More details can be found here.



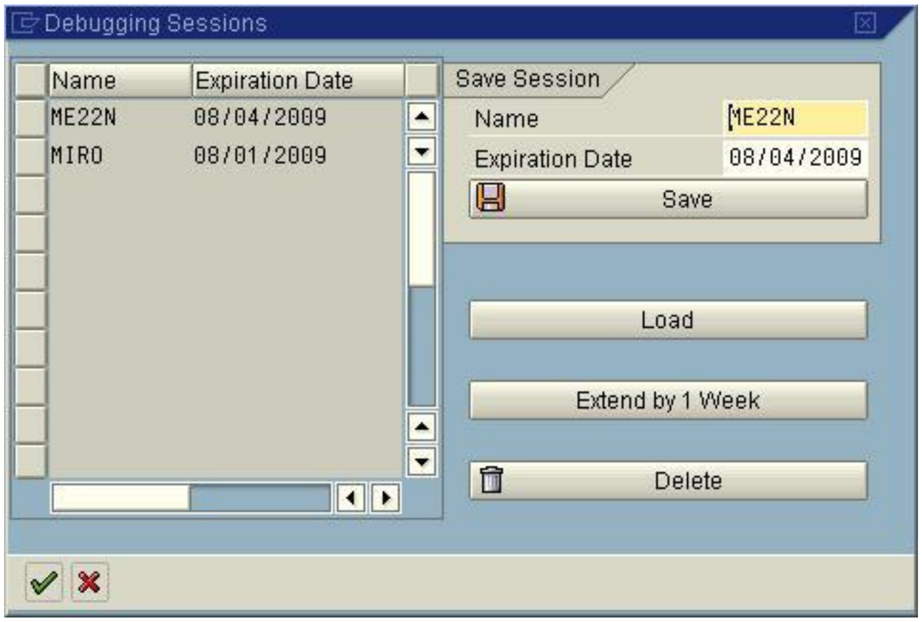
Users	EXT_USERID
☐ IP Matching	

Saving breakpoints

If you need to debug the same code again and have a set of breakpoints in place, you can create a Session to save the breakpoints and settings. Later this session can be loaded by any user, and the same set of breakpoints is restored.

<u>Path:</u> New Debugger: *Debugger -> Debugger Session -> Save*

Classic Debugger: *Debugging -> Sessions*



Name	Expiration Date
ME22N	08/04/2009
MIRO	08/01/2009

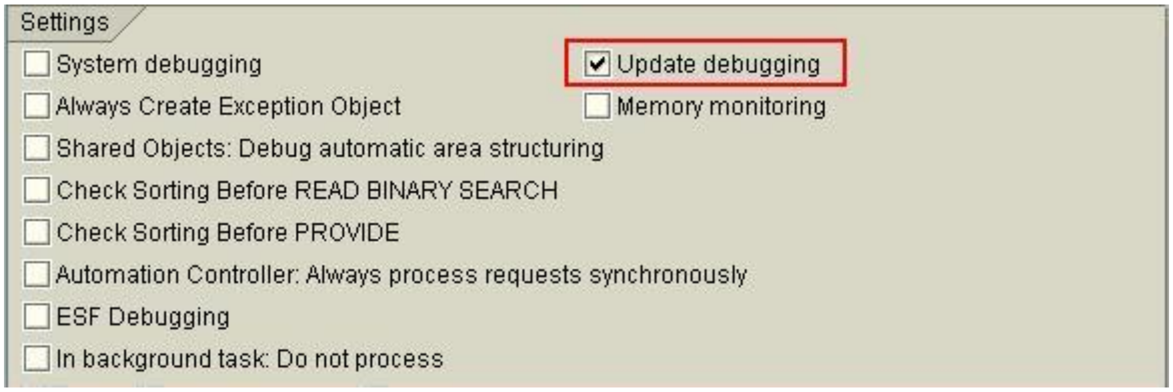
Name	ME22N
Expiration Date	08/04/2009

Note: This example uses the classic debugger.

Update debugging

We might want to debug an Update Function module, but these do not run in the same user session as the debugger, hence they cannot be debugged directly. If we enable the "Update debugging" option then a separate debugging session opens after COMMIT WORK. This is useful in debugging update terminations.

Path (classic debugger): Settings tab



Settings

- [ ] System debugging
- [✓] Update debugging
- [ ] Always Create Exception Object
- [ ] Memory monitoring
- [ ] Shared Objects: Debug automatic area structuring
- [ ] Check Sorting Before READ BINARY SEARCH
- [ ] Check Sorting Before PROVIDE
- [ ] Automation Controller: Always process requests synchronously
- [ ] ESF Debugging
- [ ] In background task: Do not process

Path (new debugger): Use the menu path: Settings -> Display/Change Debugger Setting

VMC Java debugging

When writing requirement routines / formulas using Java in the IPC (Internet Pricing Configurator), we might have to debug the IPC code. To do this follow the below steps:

1. Set the parameter **PRC_RFC** in user parameters (SU3 transaction *Parameters* tab)
2. Now execute the relevant transaction / program the parameter **PRC_RFC** makes the execution stop before an IPC Pricing call is made. Press F5 to enter into the FM, now an information message appears which says that the Java debugging is ready and mentions the port on which the debugger is waiting.
3. In another session open the transaction VMCJDB, you will see the ports which are open for debugging. Double clicking on the port will attach the debugger to the debugging session.

Detailed information can be found in SAP Help here.

Miscellaneous

There are lots of other features to explore such as:

1. Memory inspector.
2. Activatable checkpoints.
3. Performance Analysis.
4. Update management (SM13).

Related Content

Debugger functions

Remote Function Calls

IPC Debugging

For more information, visit the ABAP homepage.

Disclaimer and Liability Notice