



| z/TPF V1.1

# TPF Users Group - Fall 2009

## TPF Debugger Update

**Name:** Josh Wisniewski  
**Venue:** Development Tools  
Subcommittee

**AIM Enterprise Platform Software**  
**IBM z/Transaction Processing Facility Enterprise Edition 1.1.0**

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

# TPF Debugger Update Agenda

- Development Statement
- ALASC View
- DETAC View
- ECB Launcher
- Debug Console Command TPFMEMfill
- Watchpoint enhancements
- Memory view offset mode
- Application behavior when Debugger dumps
- Summary of other New Functionality
- Deliverable Details (APAR and TPF Toolkit level required)

## Development Statement

- A substantial amount of new function has been added to the TPF Toolkit and z/TPF Debugger in the last two years.
- We are continuing to incorporate a substantial amount of new functionality in response to requirements from both TPFUG and the FITs process into both the TPF Toolkit and the z/TPF Debugger.
- As a result, we recommend that you build into your maintenance schedules the ability to continue to upgrade to the newer versions of the TPF Toolkit and apply z/TPF Debugger APARs in order to take advantage of new functionality.

## ALASC View

- Shows all ALASC blocks for the selected ECB.
- Buttons in the upper right hand corner toggle on and off the HEX and EBCDIC panes which show the contents of the ALASC block.
- You can do a “go to address” to view the address of the ALASC block in the memory view and apply additional renderings.

The screenshot displays the ALASC View interface with three main panes:

- ALASC Summary:** A table listing ALASC blocks.
- Hex View:** A table showing the hexadecimal contents of the selected ALASC block.
- EBCDIC View:** A table showing the EBCDIC contents of the selected ALASC block.

ALASC Summary			ecbptr : 0xF000000 <Hex>			ecbptr : 0xF000000 <EBCDIC>		
Address	Module	Objc	Address	0 - 3	4 - 7	0 - 3	4 - 7	8 -
1190BEE8	QDB3	qdb3	000000001190BEE0	1190D128	1190BCA0	◀°J]	◀°µ	◀°J
1190D130	QDB2	qdb2	000000001190BEF0	80808080	80808080	0000	0000	000
			000000001190BF00	80808080	80808080	0000	0000	000
			000000001190BF10	80808080	80808080	0000	0000	000
			000000001190BF20	80808080	80808080	0000	0000	000
			000000001190BF30	80808080	80808080	0000	0000	000
			000000001190BF40	80808080	80808080	0000	0000	000
			000000001190BF50	80808080	80808080	0000	0000	000
			000000001190BF60	80808080	80808080	0000	0000	000
			000000001190BF70	80808080	80808080	0000	0000	000

## DETAC View

- DETAC view shows all DETAC blocks for the selected ECB, similar to the Data Level and DECB views.
- Buttons in the upper right hand corner allow you to toggle on and off the Data Level or DECB portions.

The screenshot shows a software window titled "DETAC" with two main sections. The top section, "Data Level DETAC Blocks", contains a table with columns: Data Lvl, Blk Addr, Blk Type, Blk Size, RID, RCC, CNC, File Addr, and File Ext. The bottom section, "DECB DETAC Blocks", contains a table with columns: DECB Addr, DECB Name, Blk Addr, Blk Type, Blk Size, RID, RCC, CNC, and File Addr. Both tables show data for four different blocks.

Data Lvl	Blk Addr	Blk Type	Blk Size	RID	RCC	CNC	File Addr	File Ext
D4	0F04EA80	0021	017D	0000	00	00	00000000	000000000000000000
D4	0F04EC00	0021	017D	0000	00	00	00000000	000000000000000000
D5	0EA91000	0031	041F	0000	00	00	00000000	000000000000000000
D6	0EA9E000	0051	0FFF	0000	00	00	00000000	000000000000000000

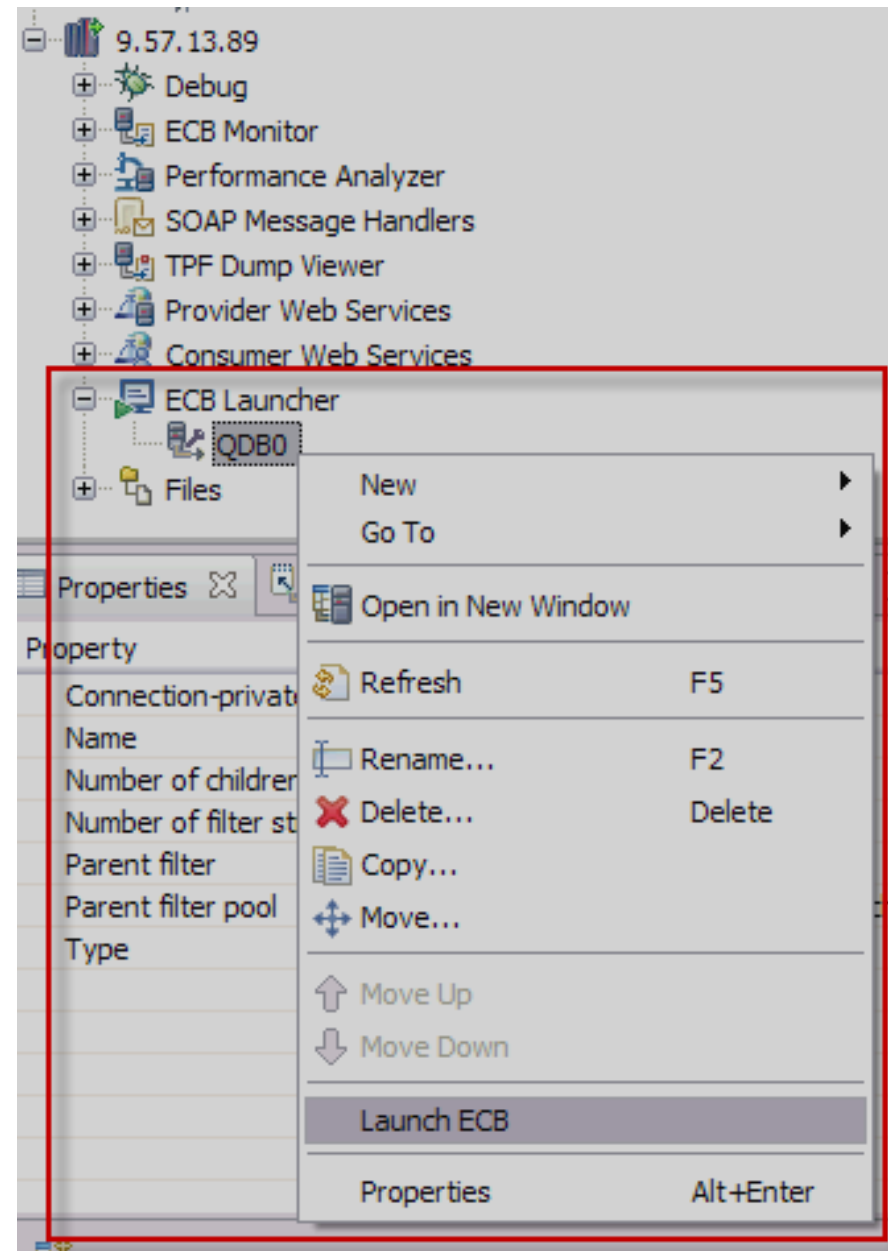
DECB Addr	DECB Name	Blk Addr	Blk Type	Blk Size	RID	RCC	CNC	File Addr
0F050CA0	QDBADECBQDBADECB	0EA92000	0021	017D	0000	00	00	00000000
0F050CA0	QDBADECBQDBADECB	0F054BE0	0031	041F	0000	00	00	00000000

## ECB Launcher

- Allows you to create a new ECB to run on z/TPF for the program you specify from the TPF Toolkit.
- The z/TPF Debugger will not automatically debug the ECBs created by the ECB Launcher. Think of the ECB Launcher as an additional testing tool that allows you execute any program for any reason. For example: generating traffic without requiring a terminal.
- If you want to debug an ECB started by the ECB Launcher, you must create a debug registration entry and register for debugging just like any other ECB. However, you will want to register using Trace by Program (LNIATA set to \*) since the application will be a created ECB.

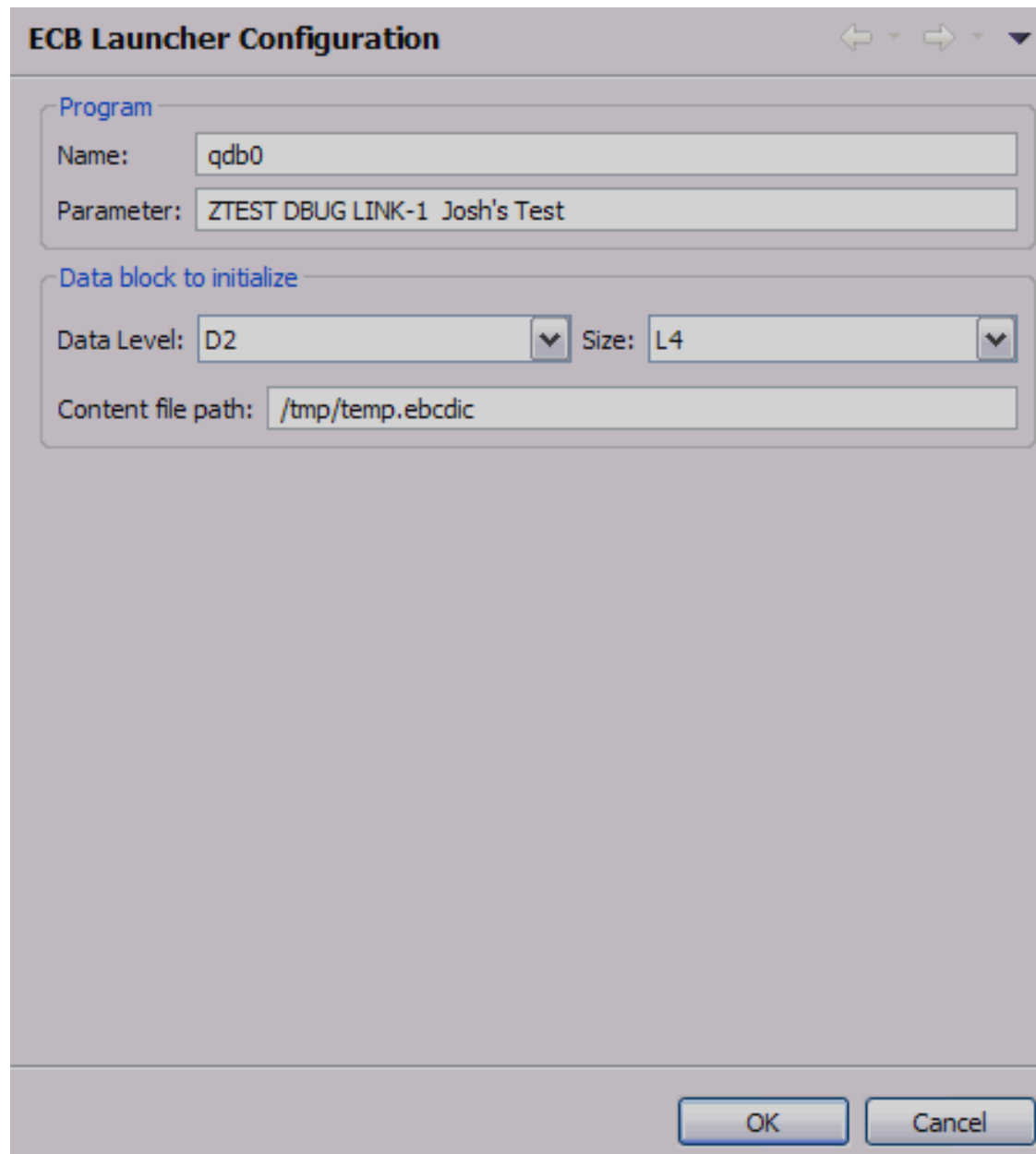
# ECB Launcher

- To use the ECB Launcher:
  - Create a new ECB Launcher entry (similar to creating a debug registration entry).
  - Specify the desired values (next slide).
  - Right click and choose “Launch ECB” to actually create the ECB on z/TPF.



# ECB Launcher

- Specify values:
  - Name – 4 character program name the created ECB will enter.
  - Parameter – text data to be copied into the ECB work area (starting at EBW000).
  - Data level to allocate and the file in the TPF file system to use to initialize the core block.



The screenshot shows the "ECB Launcher Configuration" dialog box. It has a title bar with navigation arrows. The dialog is divided into two main sections: "Program" and "Data block to initialize".

**Program section:**

- Name: qdb0
- Parameter: ZTEST DBUG LINK-1 Josh's Test

**Data block to initialize section:**

- Data Level: D2 (dropdown menu)
- Size: L4 (dropdown menu)
- Content file path: /tmp/temp.ebcdic

At the bottom right, there are "OK" and "Cancel" buttons.



# ECB Launcher

- Data Level content file formats supported (Table indicates how the files must be moved to the TPF file system):

<b>z/TPF file format</b>	<b>z/TPF file extension</b>	<b>File format on your workstation</b>	<b>FTP Type</b>
EBCDIC	.ebcdic	ASCII	ASCII
hex (EBCDIC representation of hexadecimal data)	.hex	ASCII	ASCII
binary	.bin	binary	binary
ASCII	.ascii	ASCII	binary

# ECB Launcher

- Data level file content examples:
  - EBCDIC example (any string of text like the XML here):

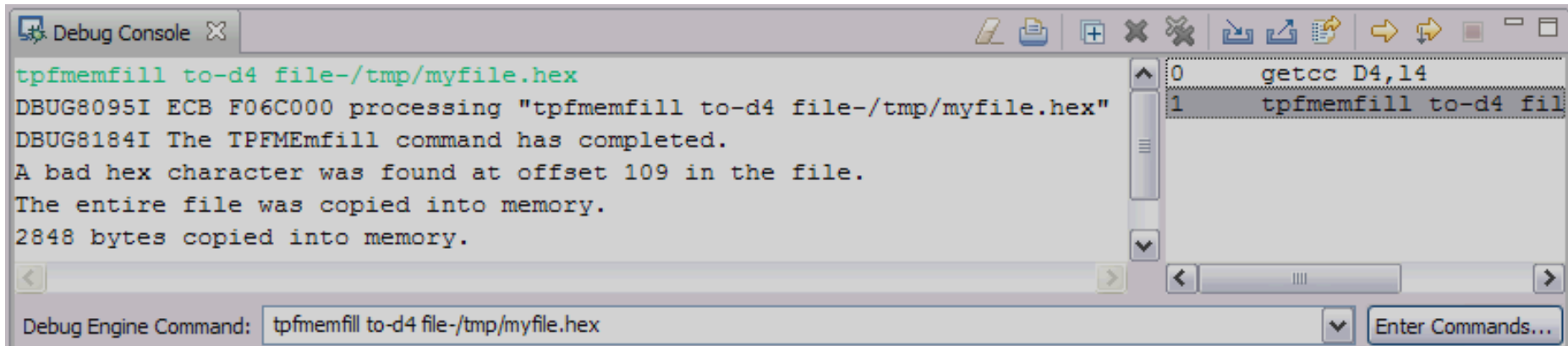
```
<?xml version="1.0" encoding="CP037"?>
  <Requests>
    <MyService RequestType="Query" Parameter="*">
      <RequestID>1</RequestID>
    </MyService>
  </Requests>
```

- hex example (data copied from a dump into a text file):

```
2A2A2A2A 2A2A2F0A 2F2A2020 20202020
4C696365 6E736564 204D6174 65726961
6C73202D 2050726F 70657274 79206F66
2049424D 20202020 20202020 20202020
20202020 20202020 20202020 202A2F0A
2F2A2020 20202020 22526573 74726963
```

## Debug Console Command TPFMEfill

- The Debug Console is a command interface to a running debugger session. Issue “help” for a list of all available commands.
- TPFMEfill is a new command that allows you to copy predefined data into memory, either from file or from another memory location.
- The TPFMEfill command is similar to the ECB Launcher data level content filling except that this command can be run any time during a debug session.



The screenshot shows a window titled "Debug Console" with a toolbar at the top. The main area contains the following text:

```
tpfmemfill to-d4 file-/tmp/myfile.hex
DBG8095I ECB F06C000 processing "tpfmemfill to-d4 file-/tmp/myfile.hex"
DBG8184I The TPFMEfill command has completed.
A bad hex character was found at offset 109 in the file.
The entire file was copied into memory.
2848 bytes copied into memory.
```

At the bottom, the "Debug Engine Command:" field contains `tpfmemfill to-d4 file-/tmp/myfile.hex` and an "Enter Commands..." button.

On the right side, there is a list of commands with a vertical scrollbar:

0	getcc D4,14
1	tpfmemfill to-d4 fil

## Debug Console Command TPFMEEmfill

- Similar to the ECB Launcher data level content fill, the debug console command TPFMEEmfill can be used to fill any memory address with the contents of a specified file.
  - TPFMEEmfill To-DX|x\* File-u\* [Maxlen-d\*]
    - To indicates the core block on a data level or hexadecimal address to copy the data to.
    - File indicates the file to copy the data from (see data level content file formats above).
    - Maxlen indicates an optional maximum length of data to copy.
    - For example:  
TPFMEEmfill T-D1 FI-/tmp/mydata.ebcdic M-100

## Debug Console Command TPFMEEmfill

- The TPFMEEmfill command can also be used to copy data from any address in the application to any address in the application.
  - TPFMEEmfill To-DX|x\* FRom-DX|x\* [Maxlen-d\*]
    - To indicates the core block on a data level or hexadecimal address to copy the data to.
    - FRom indicates the core block on a data level or hexadecimal address to copy the data from.
    - Maxlen indicates an optional maximum length of data to copy. If two hexadecimal addresses are specified, the Maxlen parameter must be specified to indicate the amount of data to move.
    - For example:  
TPFMEEmfill T-D1 FR-D5 M-100

## Memory view offset mode

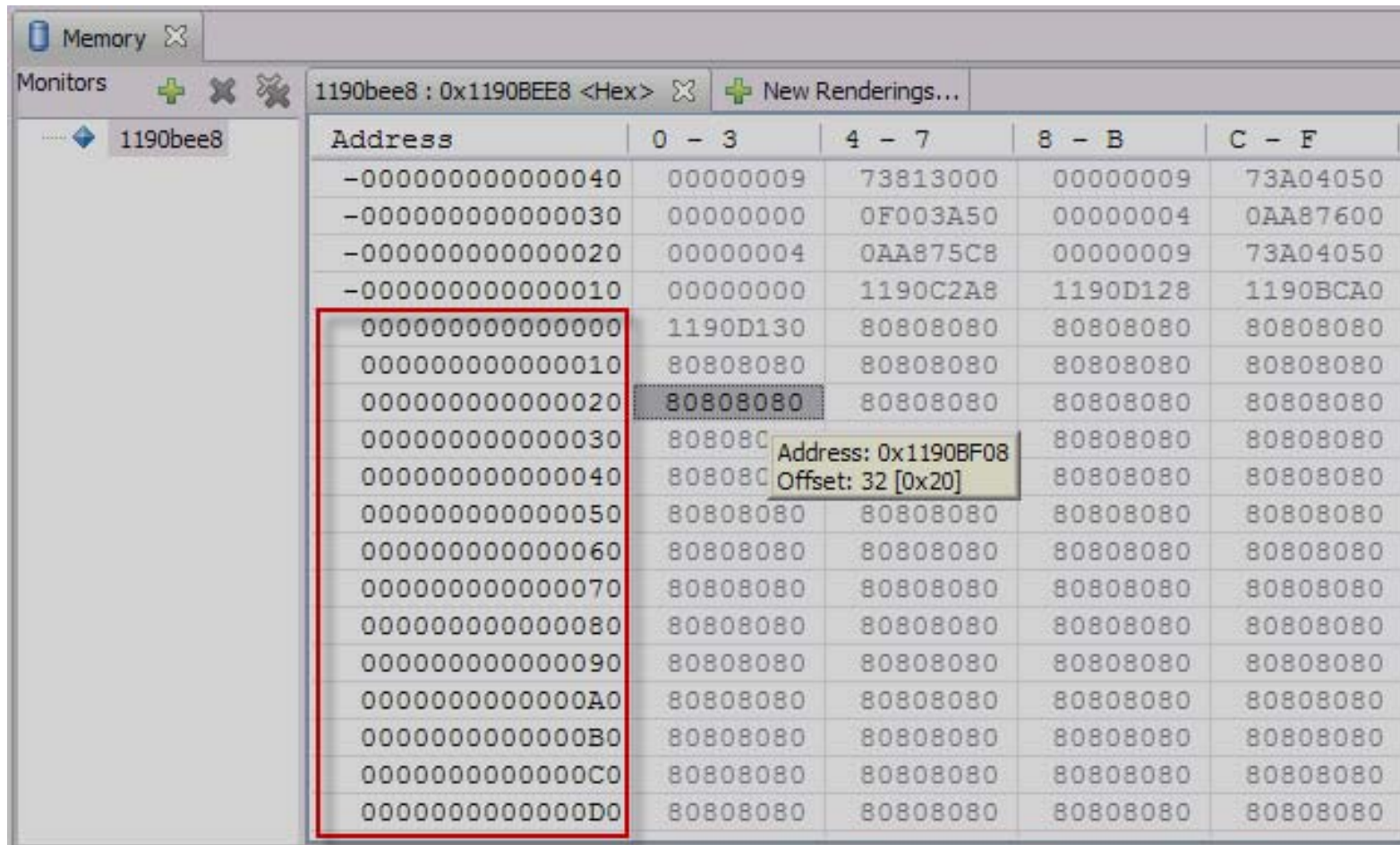
- The memory view can now be switched to show displacements from an address instead of the addresses by right clicking the address column and choosing Offset mode.

The screenshot shows the IBM Memory view interface. The main window displays a memory dump for address 1190bee8. The address column is highlighted with a red box, and a context menu is open over it. The menu options include: Add Rendering, Remove Rendering, Reset to Base Address, Go to Address..., Format..., Resize to Fit, Hide Address Column, Copy To Clipboard, Print, Properties, Dereference Pointer, and Change Display Mode. The Change Display Mode option is expanded, showing two sub-options: Address Mode (selected) and Offset Mode. A red arrow points from the Offset Mode option to the address column.

Address	0 - 3	4 - 7	8 - B	C - F
000000001190BED0	00000009	73A04050	00000000	1190C2A8
000000001190BEE0	1190D128	1190BCA0	1190D130	80808080
000000001190BEF0			80808080	80808080
000000001190BF00			80808080	80808080
000000001190BF10			80808080	80808080
000000001190BF20			80808080	80808080
000000001190BF30			80808080	80808080
000000001190BF40			80808080	80808080
000000001190BF50			80808080	80808080
000000001190BF60			80808080	80808080
000000001190BF70			80808080	80808080
000000001190BF80			80808080	80808080
000000001190BF90			80808080	80808080
000000001190BFA0			80808080	80808080
000000001190BFB0			80808080	80808080
000000001190BFC0			80808080	80808080
000000001190BFD0			80808080	80808080
000000001190BFE0			80808080	80808080

## Memory view offset mode

- Notice the Address column now shows offsets
- Hovering over a location shows the address.



The screenshot displays the IBM Memory view interface in offset mode. The main window shows a table of memory addresses and their corresponding values. The address column is highlighted with a red box, and a tooltip is visible over the address 0x1190BF08, showing the offset 32 [0x20].

Address	0 - 3	4 - 7	8 - B	C - F
-0000000000000040	00000009	73813000	00000009	73A04050
-0000000000000030	00000000	0F003A50	00000004	0AA87600
-0000000000000020	00000004	0AA875C8	00000009	73A04050
-0000000000000010	00000000	1190C2A8	1190D128	1190BCA0
0000000000000000	1190D130	80808080	80808080	80808080
0000000000000010	80808080	80808080	80808080	80808080
0000000000000020	80808080	80808080	80808080	80808080
0000000000000030	80808080	80808080	80808080	80808080
0000000000000040	80808080	80808080	80808080	80808080
0000000000000050	80808080	80808080	80808080	80808080
0000000000000060	80808080	80808080	80808080	80808080
0000000000000070	80808080	80808080	80808080	80808080
0000000000000080	80808080	80808080	80808080	80808080
0000000000000090	80808080	80808080	80808080	80808080
00000000000000A0	80808080	80808080	80808080	80808080
00000000000000B0	80808080	80808080	80808080	80808080
00000000000000C0	80808080	80808080	80808080	80808080
00000000000000D0	80808080	80808080	80808080	80808080

## Watchpoint Enhancements

- In the z/TPF debugger, watchpoints monitor for the content at a memory location to change. When a change is detected, the debugger will stop the application to show what piece of code has changed the monitored memory.
- In this example, “i” is a 4 byte integer, the value of “&i” is monitored for 4 bytes, and the application will be stopped when the value of “i” is changed to any value. For example the statement: `i = 5;`

The screenshot shows the 'Item to Watch' configuration dialog in the z/TPF debugger. The 'Address or expression' radio button is selected, and the text '&i' is entered in the adjacent field. Below this, the 'Number of bytes to watch' is set to 4. The 'Conditions on Watched Item Content' section has two unselected checkboxes: 'Stop if changed contents are equal to:' and 'Stop if instruction address is in the specified range:'. The 'Module' radio button is selected, with 'QDB0' entered in the 'Module' field. The 'Object (optional)' and 'Address range' (with 'From:' and 'To:' sub-fields) options are currently empty.



## Watchpoint Enhancements

- Watchpoints now allow you to set a value to compare the contents against.
- Building on our previous example, the application will be stopped when the value of “i” is changed to 0 from any location.

Item to Watch

Address or expression:

Register

Number of bytes to watch:

Conditions on Watched Item Content

Stop if changed contents are equal to:

Stop if instruction address is in the specified range:

Module

Module

Object (optional)

Address range

From:

To:

## Watchpoint Enhancements

- Watchpoints now allow you to specify a range in terms of Module (and object if desired) in which to perform the check.
- Building from our previous example, this watchpoint will only stop the application if the value of “i” is changed to 0 inside Module QDB0 and Object qdb0.

Item to Watch

Address or expression:

Register

Number of bytes to watch:

Conditions on Watched Item Content

Stop if changed contents are equal to:

Stop if instruction address is in the specified range:

Module

Module

Object (optional)

Address range

From:

To:

## Watchpoint Enhancements

- Watchpoints now allow you to specify a range in terms of an Address range in which to perform the check.
- Building from our previous example, this watchpoint will only stop the application if the value of "i" is changed to 0 inside the address range 0x409583fd4 to 0x40958400c.

Item to Watch

Address or expression:

Register

Number of bytes to watch:

Conditions on Watched Item Content

Stop if changed contents are equal to:

Stop if instruction address is in the specified range:

Module

Module

Object (optional)

Address range

From:

To:

## Watchpoint Enhancements

- If the expression is set to a register value, the address contained in that register will be monitored for a change as opposed to the contents of the register.
- In this example, R9 contained the address 0x1000 when the watchpoint was created, then the application will be stopped when the 4 bytes at address 0x1000 are changed to 0 in the address range 0x409583fd4 to 0x40958400c.

**Item to Watch**

Address or expression: R9

Register: R0

Number of bytes to watch: 4

**Conditions on Watched Item Content**

Stop if changed contents are equal to: 0

Stop if instruction address is in the specified range:

Module

Module: QDB0

Object (optional): qdb0

Address range

From: 409583fd4

To: 40958400c

## Watchpoint Enhancements

- Watchpoints now allow you to monitor when the contents of a register are changed. The change to value and range still apply.
- In this example, the application will be stopped when the entire contents (8 bytes) of register R9 are changed to 0 in the address range 0x409583fd4 to 0x40958400c.

Item to Watch

Address or expression: R9

Register R9

Number of bytes to watch: 8

Conditions on Watched Item Content

Stop if changed contents are equal to: 0

Stop if instruction address is in the specified range:

Module

Module: QDB0

Object (optional): qdb0

Address range

From: 409583fd4

To: 40958400c

## Application behavior when Debugger dumps

- Before PJ36657 was implemented: if the Debugger dumped, the application was pointed to EXITC processing. This could result in application dumps, terminals remaining locked and etc.
- After PJ33657 was implemented: if a Debugger dump occurs, the application is forced to take an ADB06 dump which ensures application dumps will not occur, terminals will be unlocked and etc.

## Application behavior when debugger dumps

- If the user presses the terminate button, the application is pointed to EXITC processing which can result in application dumps, terminals remaining locked and etc.
- To avoid this condition, the user can press the “Skip All Breakpoints” button in the Breakpoints view and then choose “Resume” to run the application to completion.



## Summary of other New Functionality

- The following slides summarize new functionality that was announced at the last TPFUG and are currently available.
- The slides following this summary discuss each new function in greater detail and serve as reference material.



# Summary of other New Functionality

- **Register by Function**
  - Start the debugger on an ECB when it calls a registered function.
- **Register by System Error**
  - Start the debugger on an ECB when a registered system error occurs.
- **Register by User Defined**
  - Start the debugger on an ECB virtually anywhere in the application based on the conditions registered by the user.
- **Enhanced Fork Support**
  - Register only the parent program name.
  - Enhanced stopping information and options.
- **Remote Debug Info**
  - Debug information can be automatically retrieved as needed from a remote location instead of loading all debug information to the TPF file system.

# Summary of other New Functionality

- **Add Macro Breakpoint Dialog**
  - Provides a more intuitive means to set macro and macro group breakpoints. For example, all available macro groups are shown in a drop down list.
- **ALLSVC Macro Group**
  - Stops the application at any SVC macro issued by the application.
- **TPFDF Macro Breakpoints**
  - Stops the application when a specific TPFDF macro or C/C++ macro equivalent is called by the application.
- **DFALL TPFDF Macro Group**
  - Stops the application when any TPFDF macro or C/C++ macro equivalent is called by the application.
- **Add Module**
  - Allows you to make the debugger aware of a module. This allows you to more easily set breakpoints in an unknown module.

# Summary of other New Functionality

- **ECB Summary View**
  - A quick view of the registers, work areas, data levels, and key ECB fields (which can be customized by the user).
- **Malloc View**
  - Shows all in use malloc blocks, changes in malloc usage, corrupt malloc blocks, additional information, and etc.
- **Auto-Stepping**
  - Automatically executes a step into at a given time interval.
- **System Error Retry**
  - When as debugged application generates a system error, the Debugger notifies the user. The user can now correct or avoid the error condition in the Debugger to continue debugging.
- **Trace Log Enhancement**
  - Trace log shows you all of the macros and functions executed. The debugger allows you to create the text report output on TPF without having to post process anything on Linux.

# Register By Function

- **Debugger starts when the registered ASM, C, or C++ function is entered**
- **TPF Terminal and/or condition can be specified to limit the ECBs that will start the debugger on the registered function**

**Debug Registration Session**

**Workstation Information**  
 Workstation name: \* Workstation TCP/IP address: 9.65.188.47

**TPF Terminal**  
 Terminal name: \*  
 LNIATA  IP Address  LU Name

**Registration Information**

Select a registration type: Function

Function Name: dispHelp  
 Module Name: QD\*  
 Note: Wild card in the module name may impact TPF performance or cause CTL-10  
 Object Name:

Trace created entries  
 Trace global variable initialization functions

User token:

**Condition**

ECB field or register to compare	Condition	Value to compare
	Equal to	

Limit comparison to: bytes (e.g. X'145F' for Hex, or C'test' for Char, etc.)

OK Cancel

## Register By Function

- **Wild card can be specified at the end of the module, object or function.**
- **Module can be specified as “\*” but can impact system performance and cause CTL-10 conditions**
- **Class member functions can be specified as “MyClass\*::MyGet\*”**
- **Mangled function names can be specified ie: `_ZN22IVAExceptionBreakpointC1E9IVAStrng`**
- **Conditions can be specified to test parameters passed to a function by specifying the Register to test and the value to test against.**

# Register By System Error

- **Debugger starts when the registered system error occurs**
- **TPF Terminal can be specified to limit the ECBs that will start the debugger on the registered by system error**

**Debug Registration Session**

Workstation Information  
Workstation name  Workstation TCP/IP address

TPF Terminal  
Terminal name   
 LNIATA  IP Address  LU Name

Registration Information  
Select a registration type:    
System Error Number   
Module Name   
Object Name   
 Trace created entries  
 Trace global variable initialization functions  
User token

Condition  
ECB field or register to compare  Condition   Value to compare   
 Limit comparison to:  bytes (e.g. X'145F' for Hex, or C'test' for Char, etc.)

## Register By System Error

- **Wild card can be specified for or at the end of the module and object.**
- **Debugger is only started on ECB Dumps (System dumps are not debugged).**
- **Dump number should be specified without the dump prefix and is left padded with zeros. ie OPR-I000003 can be registered as “3”.**
- **SNAPC and SERRC are supported.**

# Register by User Defined (Transaction Trapping)

- This feature allows you to start the debugger virtually anywhere based on conditions that you specify.
- Examples of types of registration:
  - Start a debugger session for a time created ECB based on the internal variable values that are of interest.

**Workstation Information**  
Workstation name \*  Workstation TCP/IP address \*

**TPF Terminal**  
Terminal name   
 LNIATA  IP Address  LU Name

**Registration Information**

Select a registration type:

ValueOf\_j

ValueOf\_j

ValueOf\_ptr

ValueOf\_something

ValueOf\_somethingelse

ValueOf\_somethingmore

Trace created entries

Trace global variable initialization functions

User token



## Register by User Defined (Transaction Trapping)

- Start a Debugger session for an ECB when it accesses a registered MQ queue by name.

**Workstation Information**


Workstation name  Workstation TCP/IP address

**TPF Terminal**

Terminal name

LNIATA  IP Address  LU Name

**Registration Information**

Select a registration type:  

Name of Queue Accessed

Trace created entries

Trace global variable initialization functions

User token

- See Appendix A for more information

## Register by User Defined (Transaction Trapping)

- CTEST now uses the User Defined Registration support. Code ctest() in your application and then register with the new IBM\_CTEST registration type.

**Workstation Information**

Workstation name  Workstation TCP/IP address

**TPF Terminal**

Terminal name

LNIATA  IP Address  LU Name

**Registration Information**

Select a registration type:

Trace created entries

Trace global variable initialization functions

User token

## Appendix A: Register by User Defined

- **How is register by user defined setup by an administrator?**
  1. An XML file on the workstation defines the registration type and parameters that a user would register
  2. Code a user exit function or 4 character program to evaluate the registered conditions
  3. Add a call to the application code to the registration handler.
    - Performance sensitive macros are provided such that this code can be left in production code but avoid the registration handler code and have minimal effect on performance.
    - Assembler and C/C++ interfaces provided.
    - See the source segments `c_udrt.h`, `udrpc.mac`, `iudrt.mac`, `cudrt.c`, `cdbxud.c` and `cdbx.c` for more information and examples. Or search the TPF Toolkit help for the topic “custom defined registration”.
- **The following slides show an example of a user defined registration for a time initiated application QDB0 based on internal variable values.**

## Appendix A: Register by User Defined

### 1. Modify the file <TPF Toolkit install dir>\Config\TPFSHARE\Debug Registration\customDebugRegTypes.xml

- Ids 101-255 are for customer use (0-100 are reserved for IBM)
- Specify the registration name and up to 6 parameter names

```
<customRegistration>  
  <id>101</id>  
  <name>MQByQueueName</name>  
  <parameter>Name of Queue Accessed</parameter>  
</customRegistration>
```

```
<customRegistration>  
  <id>102</id>  
  <name>TimeCreatedQDB0</name>  
  <parameter>ValueOf_i</parameter>  
  <parameter>ValueOf_j</parameter>  
  <parameter>ValueOf_ptr</parameter>  
  <parameter>ValueOf_something</parameter>  
  <parameter>ValueOf_something_else</parameter>  
  <parameter>ValueOf_something_more</parameter>  
</customRegistration>
```

### 2. Restart the TPF Toolkit

## Appendix A: Register by User Defined

### 3. Implement the resolving function to test application state against the user registered values

```
unsigned int CDBX_TimeCreatedQDBOCheck(struct tpf_UserDefRegTypStruct* ptr, struct itbpentry* reg)
{
    unsigned rc = FALSE; //set default return to false

    switch(ptr->udrt_id)
    {
        case 102:
        {
            //verify that i matches
            if(*((unsigned int *)ptr->udrt_parm1) != atoi((char*)reg->itbp_udrt_parmValue[0]))
                break; //no, we're done
            //verify that j matches
            if(*((unsigned int *)ptr->udrt_parm2) != atoi((char*)reg->itbp_udrt_parmValue[1]))
                break; //no, we're done
            //verify that ptr matches
            if(strcmp((char*)ptr->udrt_parm3, (char*)reg->itbp_udrt_parmValue[2]) != 0)
                break; //no, we're done

            //passed all tests, start the debugger
            rc = TRUE;
            break;
        }
        case 103:
            //...
        default:
            break;
    }
    return rc;
}
```

## Appendix A: Register by User Defined

4. Update the application code to call User Defined Registration handler, passing in the resolving function to use.

C++ qdb0.cpp

```
char * sys_state = (char *) cinfc_fast(CINFC_CMMSTI);

if(tpf_UserDefRegTypPerfCheck(102))
{
    struct tpf_UserDefRegTypStruct temp = {0};
    temp.udrt_id = 102;
    temp.udrt_funcptr = (tpf_UserDefRegTypUserExit *)CDBX_TimeCreatedQDB0Check;
    temp.udrt_parm1 = (void*)&i;
    temp.udrt_parm2 = (void*)&j;
    temp.udrt_parm3 = ptr;
    tpf_UserDefRegTypHandler(&temp);
}
```

## Appendix A: Register by User Defined

### 5. Register the debugger with the conditions to start the debugger on the application

**Workstation Information**

Workstation name \*  Workstation TCP/IP address \*

**TPF Terminal**

Terminal name

LNIATA  IP Address  LU Name

**Registration Information**

Select a registration type:  ▼

ValueOf\_i

ValueOf\_j

ValueOf\_ptr

ValueOf\_something

ValueOf\_somethingelse

ValueOf\_somethingmore

Trace created entries

Trace global variable initialization functions

User token

## Appendix A: Register by User Defined

- 6. Start the application to be debugged.**
- 7. When the application is started, the `tpf_UserDefRegTypHandler` function will call the resolving function passed to it, to test the application state against each registration entry of the same type.**
- 8. If the resolving function returns `TRUE`, the Debugger will start at the next executable line of debuggable code.**



# Enhanced Fork Support

- “Trace created entries” must be checked.
- Register only the Parent program (Child program does not need to be registered any longer)
- Fork is generic term for CREMC, CREDC, SWISC create, TPF\_fork, etc.

**Debug Registration Session**

**Workstation Information**  
 Workstation name: jwisnie      Workstation TCP/IP address: \*

**TPF Terminal**  
 Terminal name: \*  
 LNIATA     IP Address     LU Name

**Registration Information**  
 Select a registration type: Program

QDB0      Add      Remove

Trace created entries  
 Trace global variable initialization functions

User token: \_\_\_\_\_

**Condition**

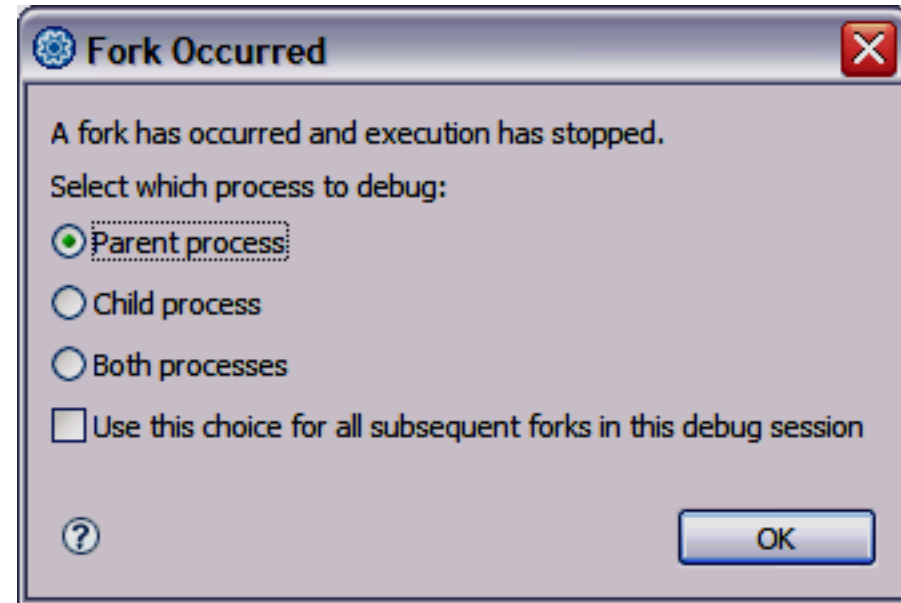
ECB field or register to compare	Condition	Value to compare
_____	Equal to	_____

Limit comparison to: \_\_\_\_\_ bytes      (e.g. X'145F' for Hex, or C'test' for Char, etc.)

OK      Cancel

# Enhanced Fork Support

- **Three options available when a fork event occurs:**
  - **Parent Process:** Means ignore the fork event. The Parent continues the previous execute request (step into, run, etc).
  - **Child Process:** Means create a new debugger session for the Child process. The Parent continues the previous execute request (step into, run, etc).
  - **Both Processes:** Means create a new debugger session for the Child process. The Parent process stops at the next executable line in debuggable code.



## Enhanced Fork Support

- **When the “Fork Occurred” dialog appears, the Debug Console now shows a message indicating where the fork type event occurred and what program the created ECB will enter.**

```
DEBUG8158I Module QDB0 issued a tpf_fork() to QDBM  
in object qdb0go.o, offset 0x1B6, function goCases
```

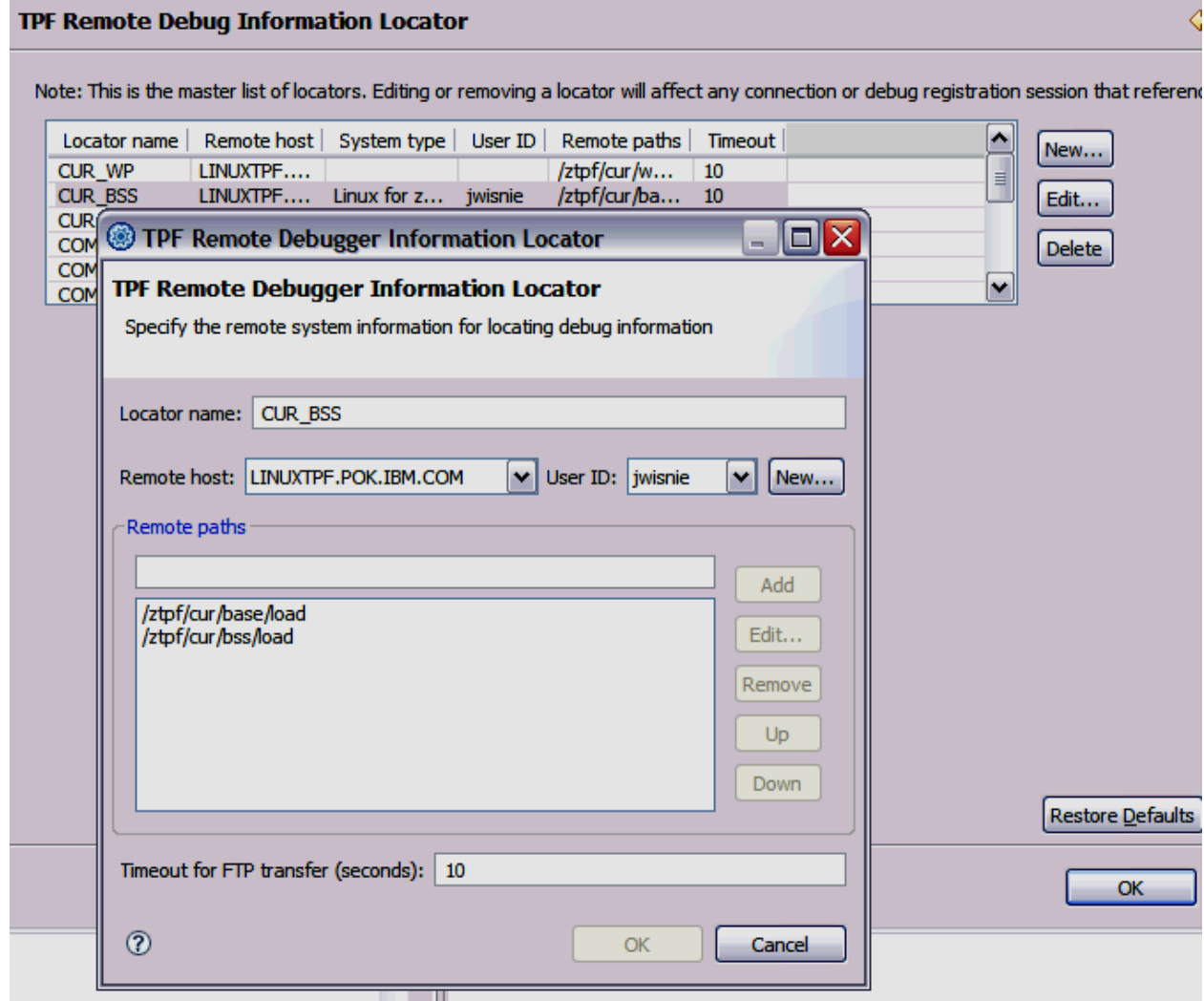
## Remote Debug Info

- **Allows you to store your z/TPF debug information files somewhere other than on the TPF file system. However, loading debug information via OLD or TLD is still preferred as it will ensure that the debug information matches the loaded code.**
- **The Debugger detects when a debug information file is not loaded and attempts to FTP the debug information from the remote location.**
- **Multiple FTP paths can be specified but to receive the best performance we recommend 3 or less FTP paths.**
- **Version codes in the PAT are used to find a match on the remote system.**
- **FTPed debug information has the dbgftp suffix. For Example module ABCD with version code ZZ would be FTPed to /tpfdbgelf/ab/abcd/ABCDZZ.dbgftp**
- **FTPed debug information will be deleted if the debug information is loaded by OLD or TLD.**

## Remote Debug Info

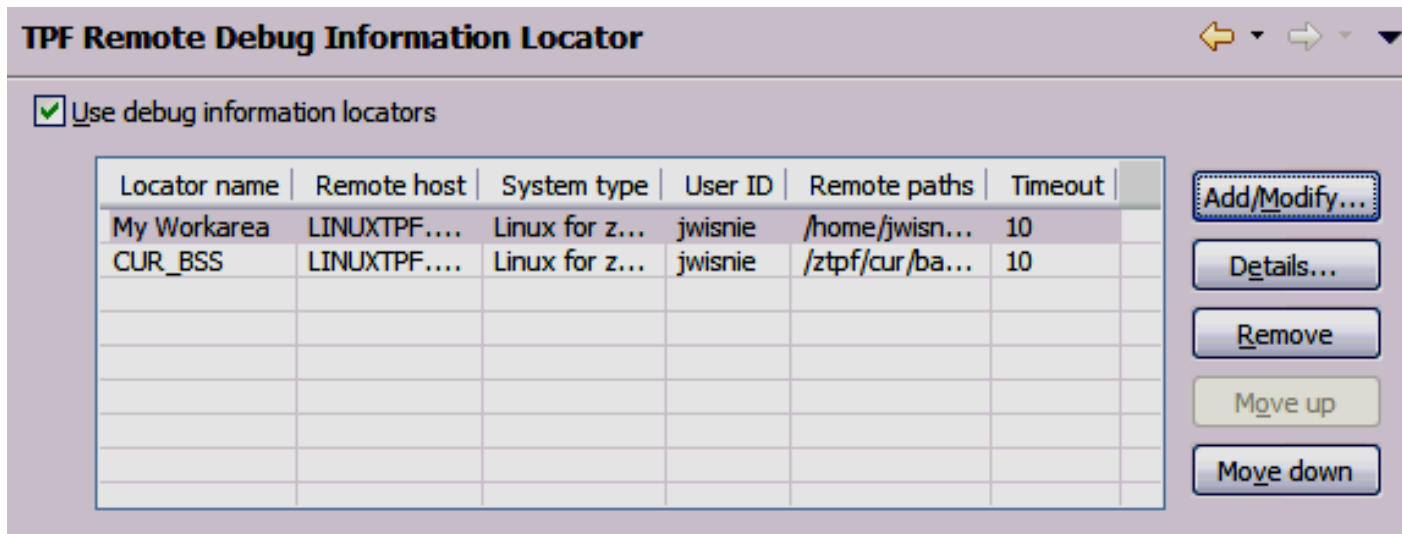
To use the Remote debug info feature

1. Create the “locators” from the menu option Windows-> Preferences-> Run/Debug->TPF Remote Debug Information Locator. Locators specify the Remote Host name, Fully qualified path, User Id, Password, and time out value.



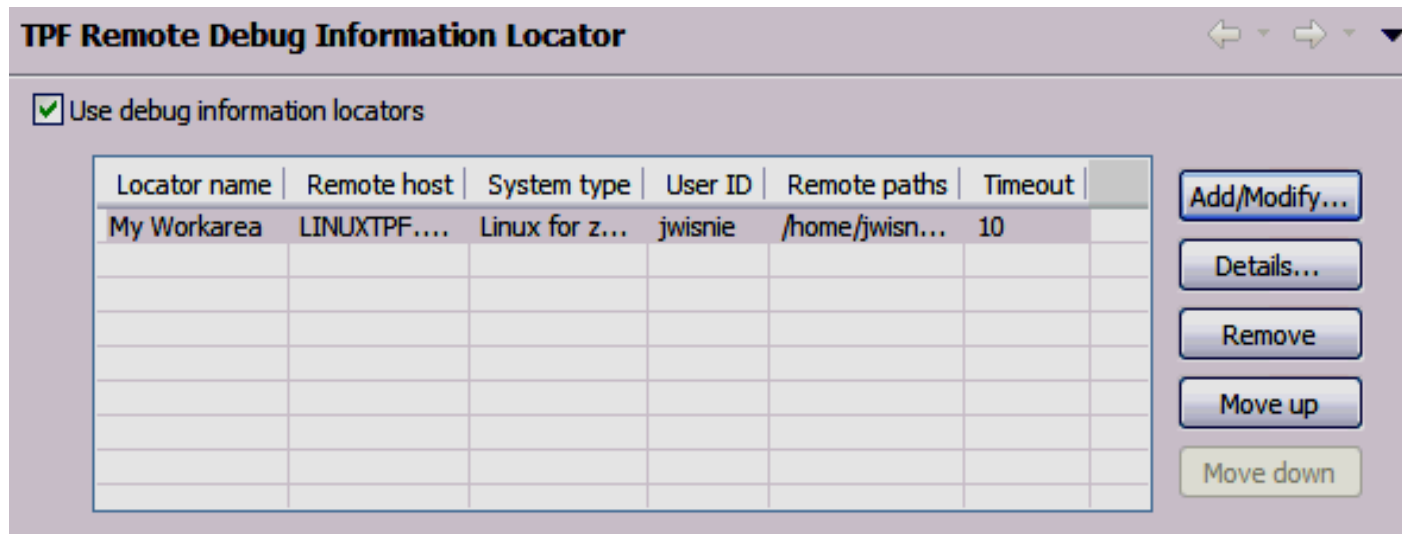
## Remote Debug Info (TPF Connection)

2. Right click the TPF Connection from the RSE and choose properties. Add the locators in the search order desired. These locators will be used by default for the debug sessions, dump viewer, and ECB monitor.



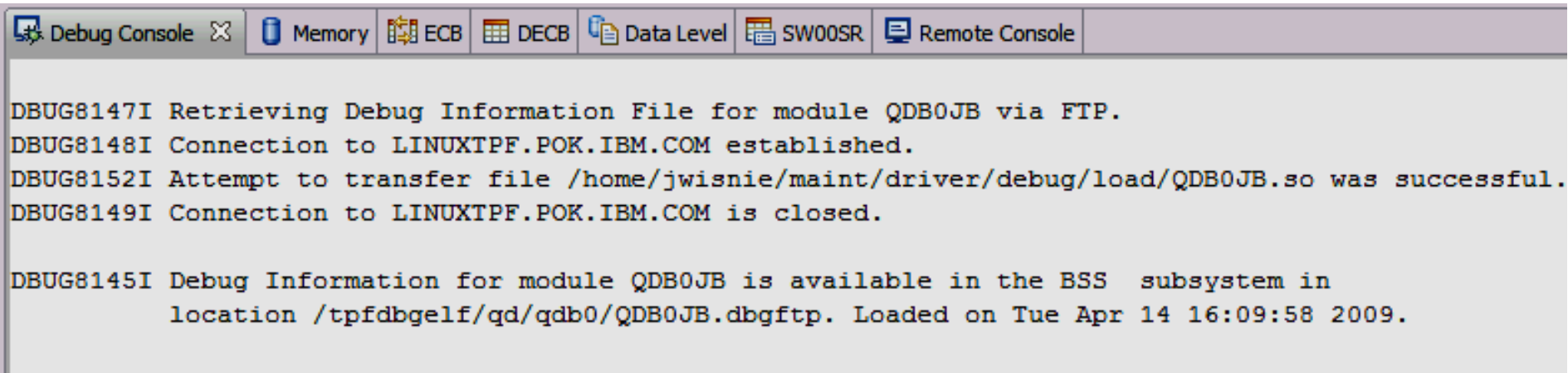
## Remote Debug Info (Debug Session)

- The locators can be customized for each Debug Session regardless of the settings at the Connection level. Right click the Debug Session from the RSE and choose properties. Add the locators in the search order desired.**



## Remote Debug Info (Debug Session)

- **Debug console messages are now sent to the TPF Toolkit to notify the user if debug information could be located and what debug information file was used.**



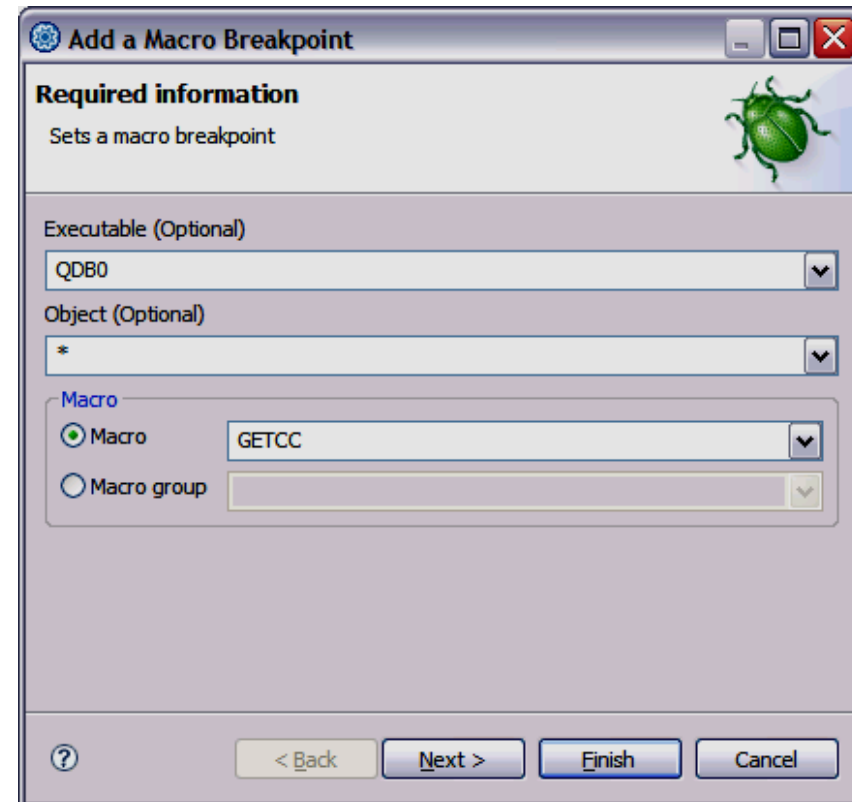
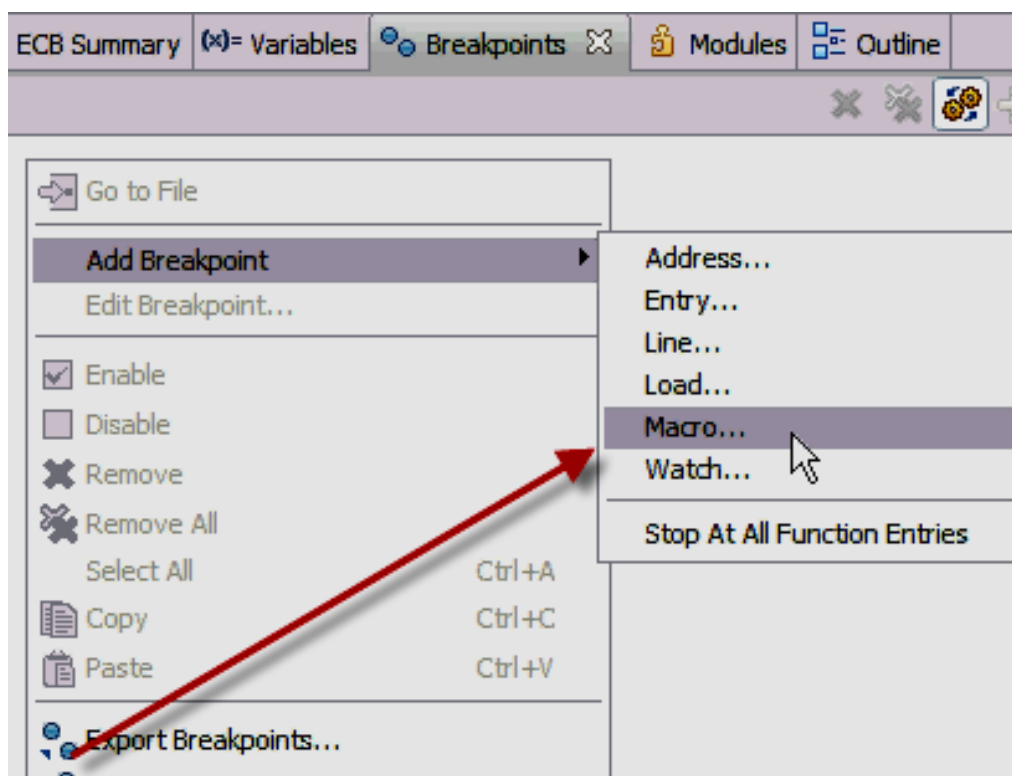
```
Debug Console x Memory ECB DECB Data Level SW00SR Remote Console
DEBUG8147I Retrieving Debug Information File for module QDB0JB via FTP.
DEBUG8148I Connection to LINUXTPF.POK.IBM.COM established.
DEBUG8152I Attempt to transfer file /home/jwisnie/maint/driver/debug/load/QDB0JB.so was successful.
DEBUG8149I Connection to LINUXTPF.POK.IBM.COM is closed.

DEBUG8145I Debug Information for module QDB0JB is available in the BSS subsystem in
           location /tpfdbgelf/qd/qdb0/QDB0JB.dbgftp. Loaded on Tue Apr 14 16:09:58 2009.
```



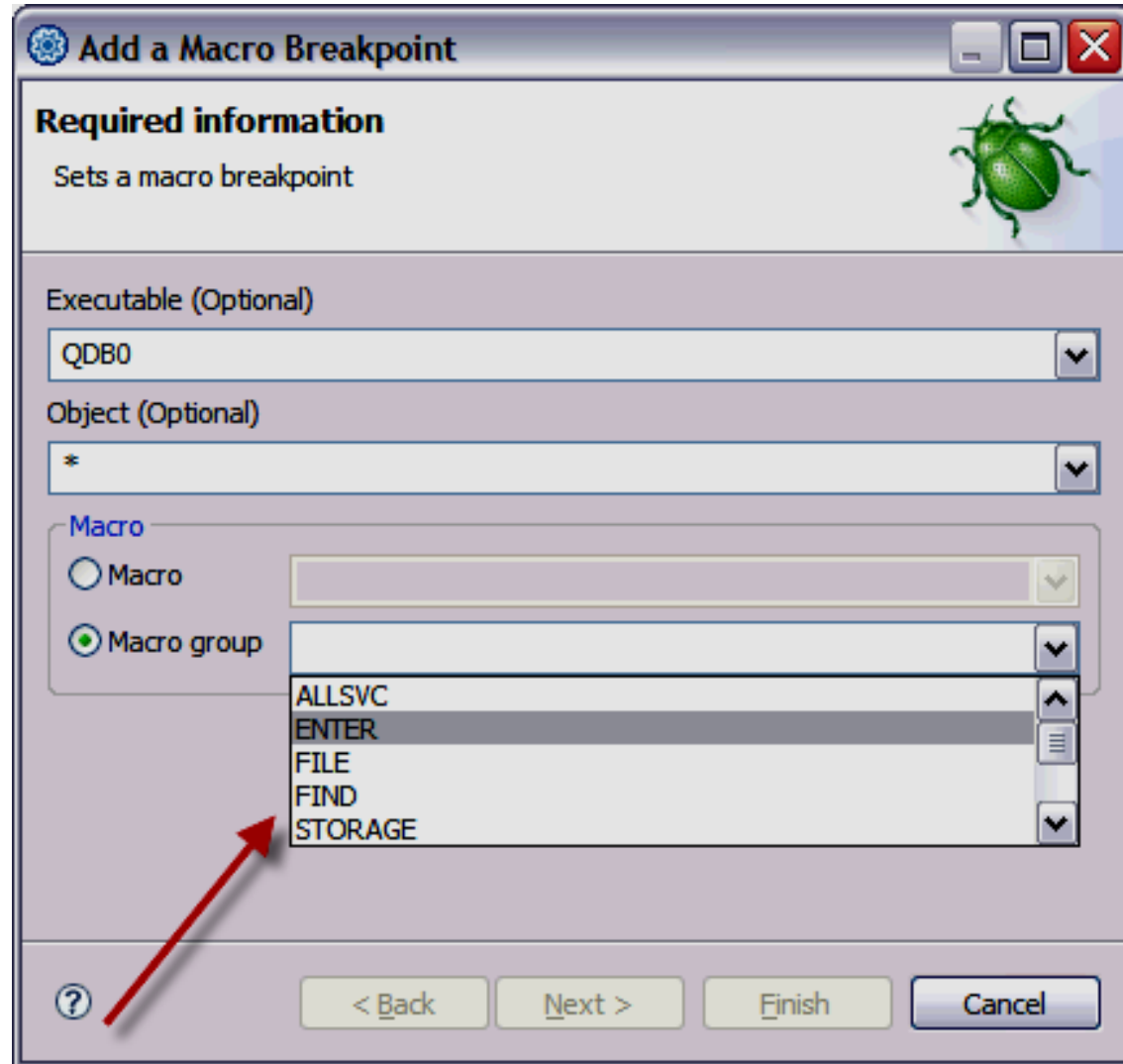
## Add Macro Breakpoint

- **Macro Breakpoints now have their own dialog box which is available by right clicking in the breakpoint view (choosing Entry breakpoint and Defer is no longer required).**



## Macro Group List

- **Clicking the drop down arrow provides a list of all available Macro Groups.**



# ALLSVC Macro Group

- **The ALLSVC Macro Group will stop the application for All SVC type Macros**

**Add a Macro Breakpoint**

**Required information**  
Sets a macro breakpoint

Executable (Optional)  
QDB0

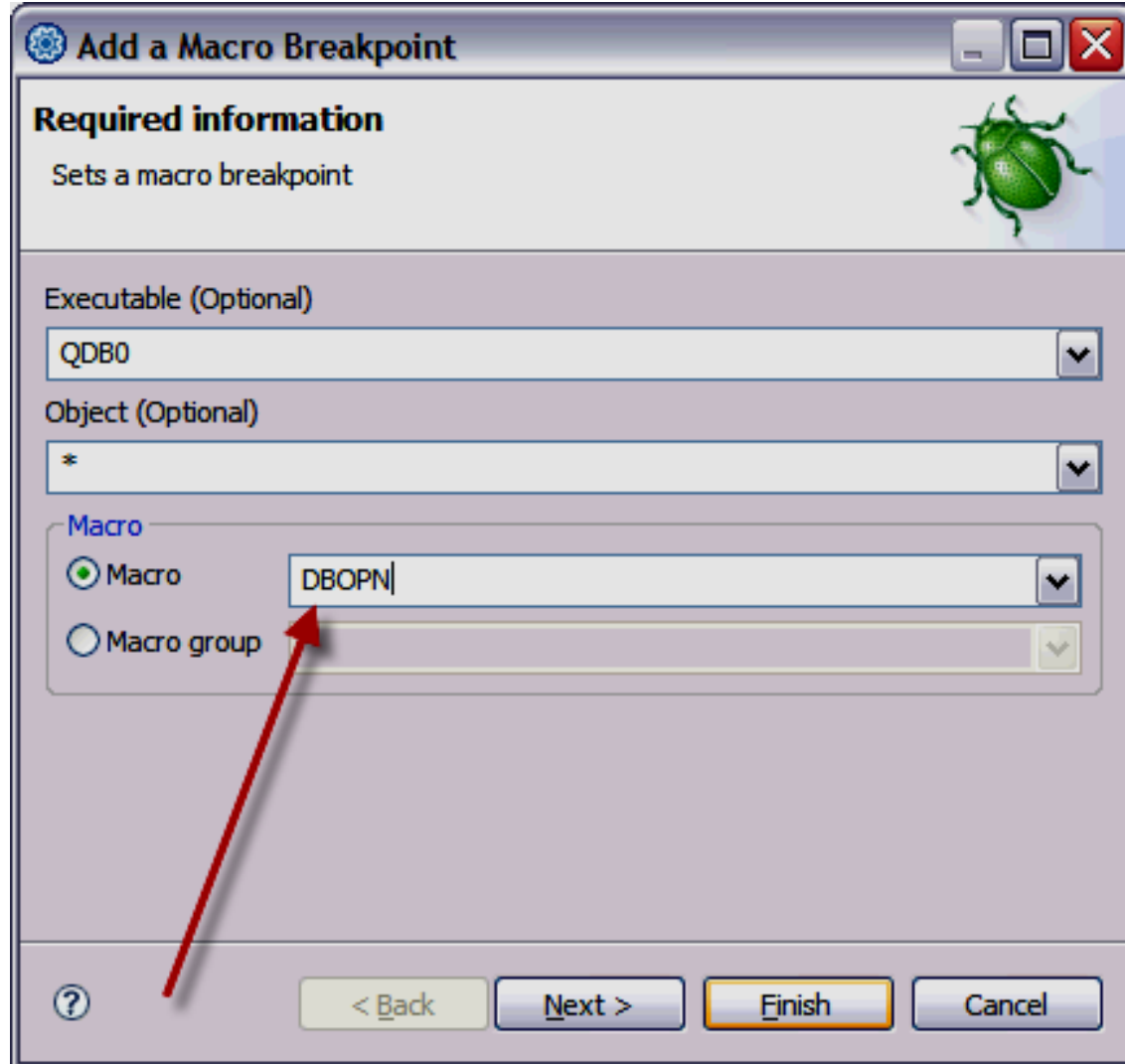
Object (Optional)  
\*

Macro  
 Macro  
 Macro group ALLSVC

< Back   Next >   Finish   Cancel

# TPFDF Macro Breakpoints

- **TPFDF Macro Names can now be entered through the Macro Breakpoint pane.**



**Add a Macro Breakpoint**

**Required information**  
Sets a macro breakpoint

Executable (Optional)  
QDB0

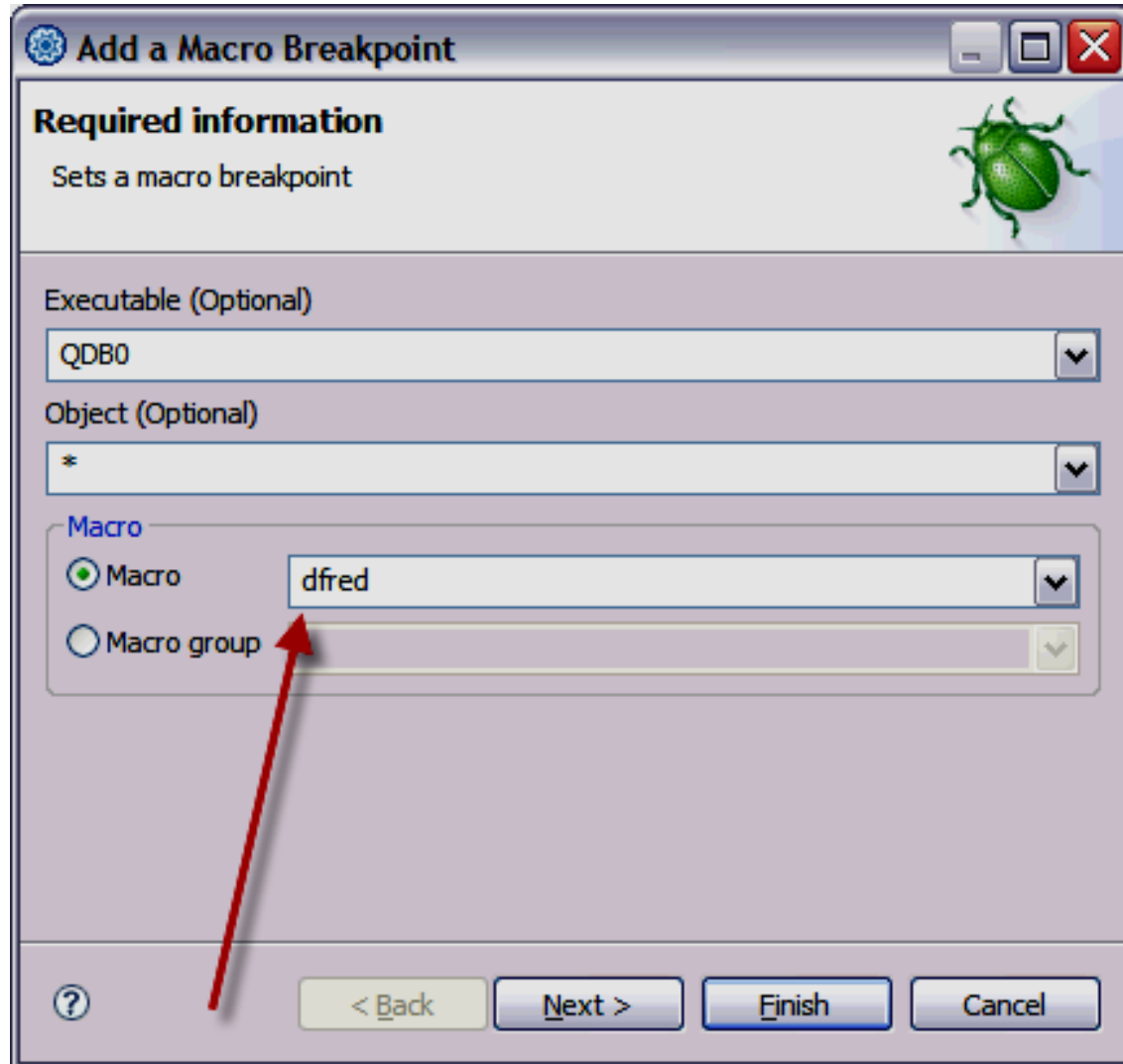
Object (Optional)  
\*

Macro  
 Macro DBOPN  
 Macro group

? < Back Next > Finish Cancel

# TPFDF Macro Breakpoints

- **TPFDF C/C++ Macro equivalents can now be entered through the Macro Breakpoint pane (ie. dfred, dfopn, etc).**



**Add a Macro Breakpoint**

**Required information**  
Sets a macro breakpoint

Executable (Optional)  
QDB0

Object (Optional)  
\*

Macro  
 Macro dfred  
 Macro group

? < Back Next > Finish Cancel

# DFALL TPFDF Macro Group

- **The DFALL Macro Group will stop the application when any TPFDF Macro or C/C++ Macro equivalents is executed by the application.**

**Add a Macro Breakpoint**

**Required information**  
Sets a macro breakpoint

Executable (Optional)  
QDB0

Object (Optional)  
\*

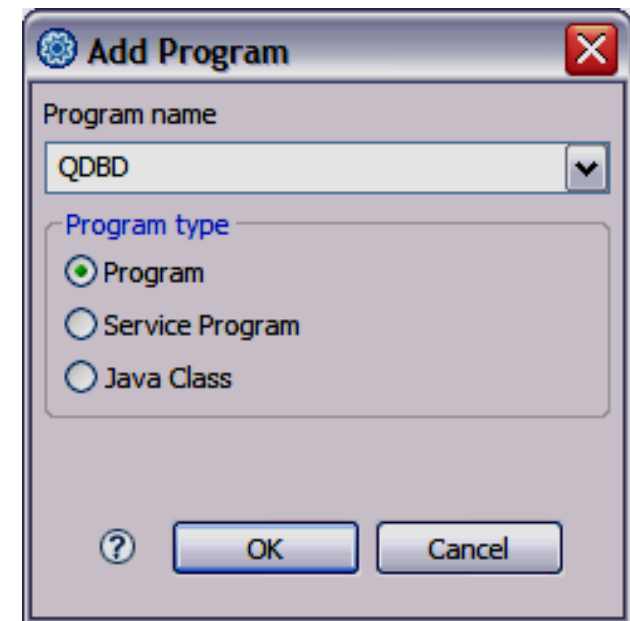
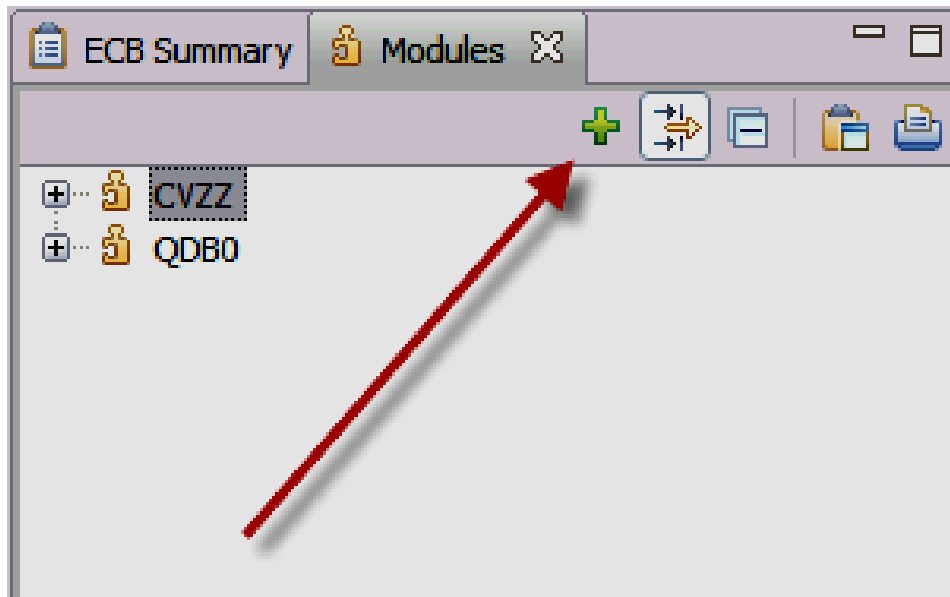
Macro  
 Macro  
 Macro group

DFALL

< Back   Next >   Finish   Cancel

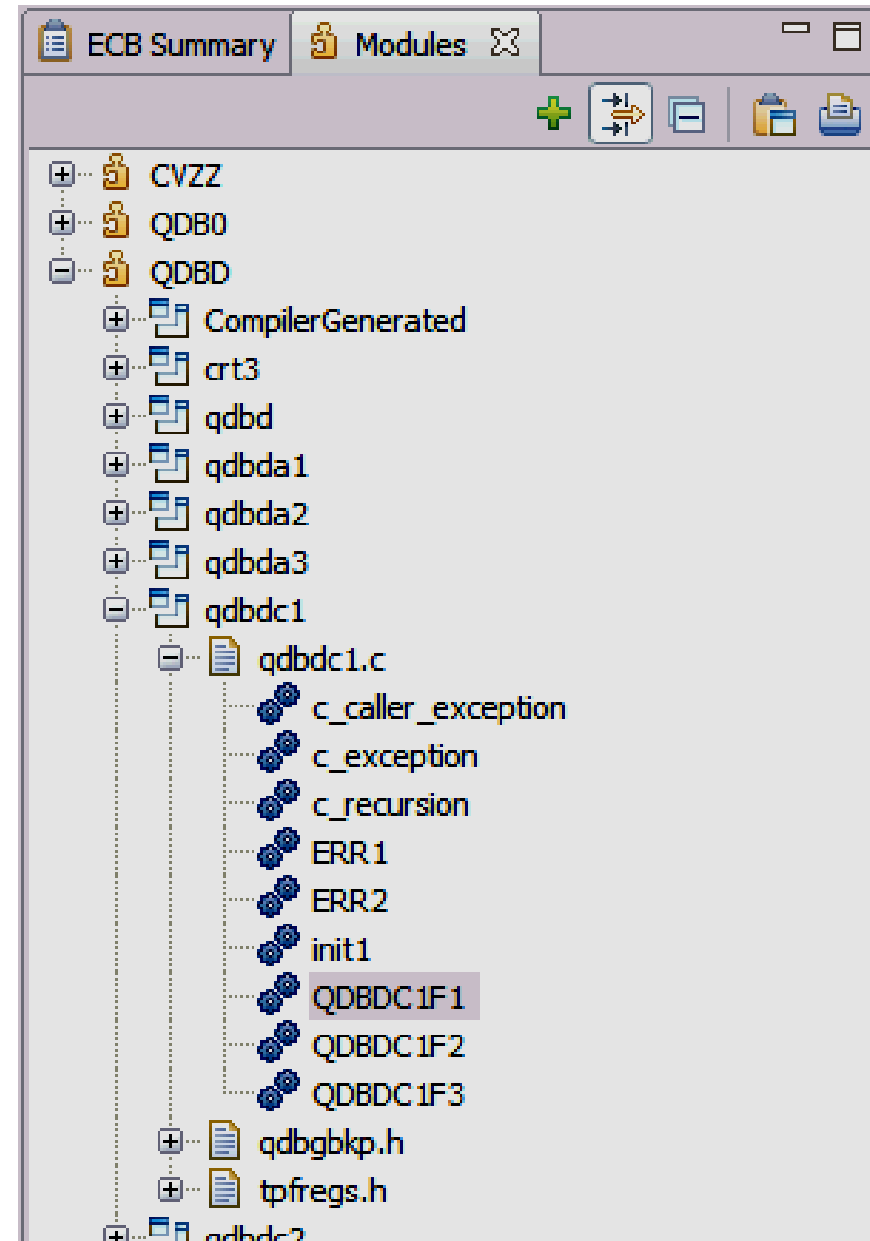
## Add Module (Add Breakpoint in any Module)

- **Add Module** allows the user to make the debugger aware of a module that has not been debugged or appeared on the stack.
- From the **Modules** view, you can now choose the **Green Plus** (Add program to debug). From the **Add Program** dialog, enter the 4 character program name and choose **OK**.
- The debugger will attempt to get debug information for the specified module and allow you to perform a variety of actions



## Add Module (Add Breakpoint in any Module)

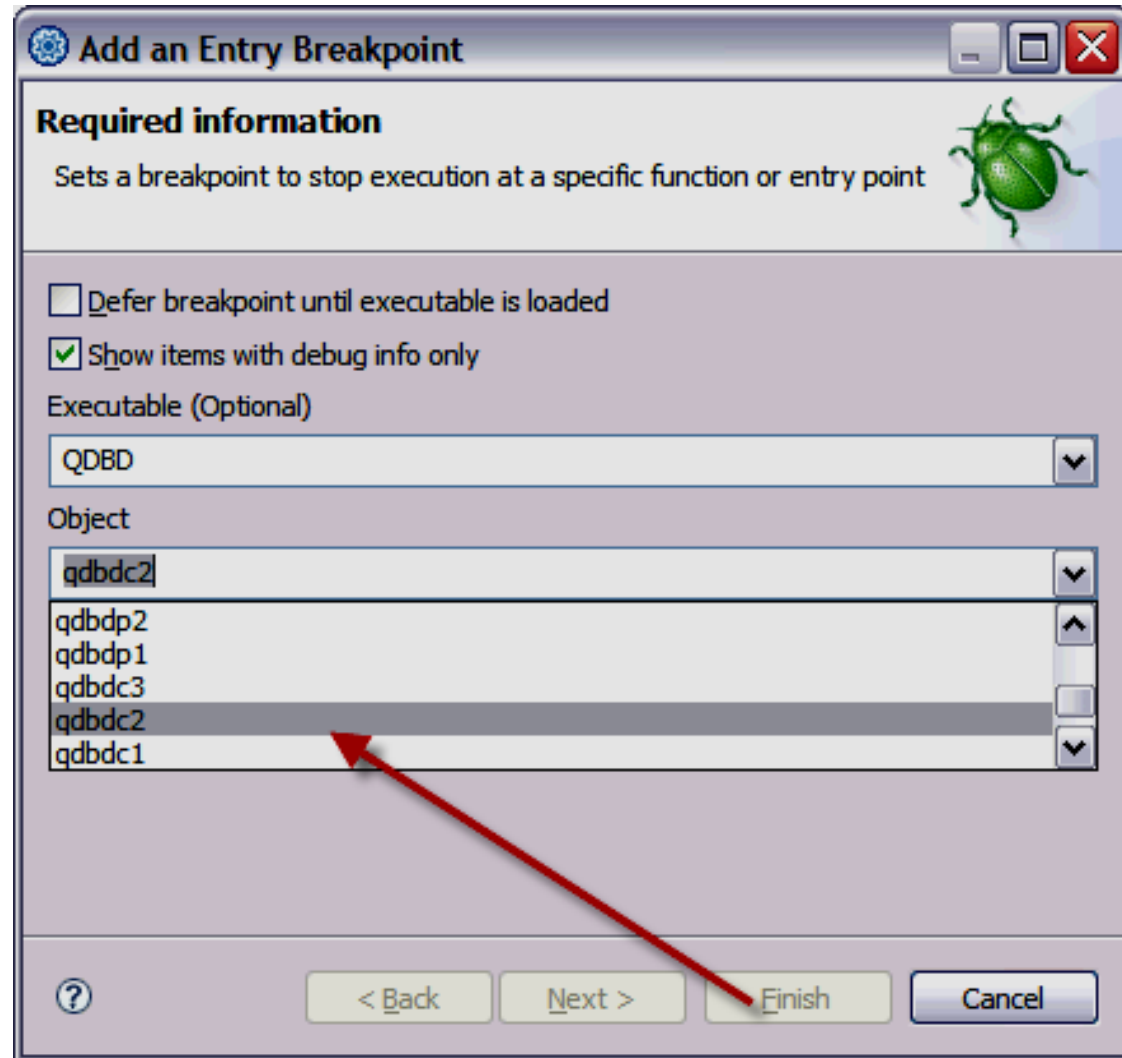
- **Available actions in the Modules view**
  - See all objects in a Module
  - See all files compiled to create an object
  - See all functions in a file
  - Open the file with a double click on the file or function (then set line/address breakpoints from a double click in the source)
  - Right function to set function breakpoints





## Add Module (Add Breakpoint in any Module)

- **Actions available from the breakpoints view**
  - When adding a breakpoint, the executable, objects, and functions will be available from the drop down lists for the added module.



# ECB Summary View

- Quick view of common ECB areas
- Backed by XML for easy customization
- Individual panes can be toggled on and off
- Control and floating point registers are available at right click of the registers pane

The screenshot shows the ECB Summary View interface with the following panes:

**Registers:**

R0	0000000000000010	R1	0000000300000000	R2	0000000000000000
R3	0000000000000000	R4	00000000D8C4C2F0	R5	00000003982A4178
R6	00000003973BC1E8	R7	00000000DDF0000	R8	00000000DDF00C5
R9	0000000000000000	R10	0000000000000000	R11	00000000DD0F430
R12	00000003973BB000	R13	00000003979DF8F8	R14	00000000013B75E
R15	00000000DD0F430				
PSW	4715000180000000		00000003979D509C		

**Work Area:**

W00	C4C2E4C7	004	C3E5E9E9	008	80B00000	012	00000000
016	00000000	020	00000000	024	00000000	028	00000000
032	00000000	036	00000000	040	01000000	044	00000000
048	00000000	052	E2D4D7C2	056	010000C2	060	80B00000
064	00000000	068	00000000	072	00008400	076	04000000
080	E3C5E2E3	084	00000000	088	00000000	092	036DD8C8
096	00000000	100	00000000	SW1	00000000	CM1	01000000

**Miscellaneous:**

FAP	00FF00002C05802D	GLA	0240A000	HLD	00
ACN	00000002	SUI	00	SSU	FF00
ISN	0001	CPD	B	GLY	02412000
IOC	0001	OUT	010000	DET	0B400288
PAT	00000000ACC8018				

**Data Level:**

Name	CE1FAx	CE1FMx	CE1CRx	CE1CTx	CE1CCx	SUD	DCT
D0	00000000	00000000	0B406E80	0021	017D	00	00
D1	00000000	00000000	0B408000	0001	0FFF	00	00
D2	00000000	00000000	00000000	0001	0000	00	00
D3	00000000	00000000	00000000	0001	0000	00	00

## Malloc View

- The malloc view is made up of 4 panes which can be individually hidden by the buttons in the upper right corner of the view.

The screenshot shows the TPF Malloc View interface with the following data:

**Changed Blocks**

ADDR	LEN	APGM	RPGM	In use	Corrupt
119F6000	7D8	QDB0		yes	no
119F8000	7D8	QDB0	QDB0	no	no

**In Use Blocks**

ADDR	LEN	APGM	NAME
119F6000	7D8	QDB0	1stQDB0 Malloc
119F7000	858	CJ00	
119F1300	70	CJ00	
119F1000	48	CFVZ	

**Freed Blocks**

ADDR	LEN	APGM	RPGM
119F4C00	2D8	CFVZ	CFVZ
119F5000	258	CFVZ	CFVZ
119F8000	7D8	QDB0	QDB0
11A059F0	2070	CJ00	CJ00

**Selected Block**

```

Address      119F8000
Size (user)  7D8
Size (real)  1518
Name
Corrupted    No
State        Free
Heapcheck    No
ECB SVA      F04E000
Thread id    0
Allocating   Program
              Address  409B3637E
              Module   QDB0JB
              Object   qdb0.cpp
              Function  QDB0
Freeing      Program
              Address  40ABFBB1C
              Module   CPP1
              Object   del_op.cc
              Function  _ZdlPv
  
```

## Malloc View

- The inuse and free panes shows the malloc blocks that are inuse or free respectively
- The changed panes show the changes in malloc since the last refresh

Changed Blocks					
ADDR	LEN	APGM	RPGM	In use	Corrupted
119F6000	7D8	QDB0		yes	no
119F8000	7D8	QDB0	QDB0	no	no

In Use Blocks				
ADDR	LEN	APGM	NAME	
119F6000	7D8	QDB0	1stQDB0 Malloc	
119F7000	858	CJ00		
119F1300	70	CJ00		
11A00000	4038	CJ00		
119F3400	1B0	CJ00		
119F3800	170	CJ00		
119F3000	130	CJ00		

## Malloc View

- The selected block pane shows additional information about a malloc block that is selected in one of the other panes such as the program that did the malloc or free.

The screenshot shows a window titled "Selected Block" with a list of attributes and their values. The attributes are listed in a monospaced font. The values are right-aligned. The window has a scrollbar on the right and navigation arrows at the bottom.

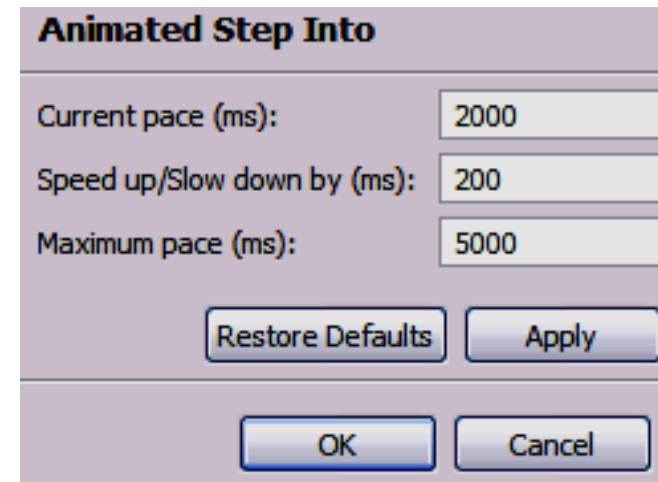
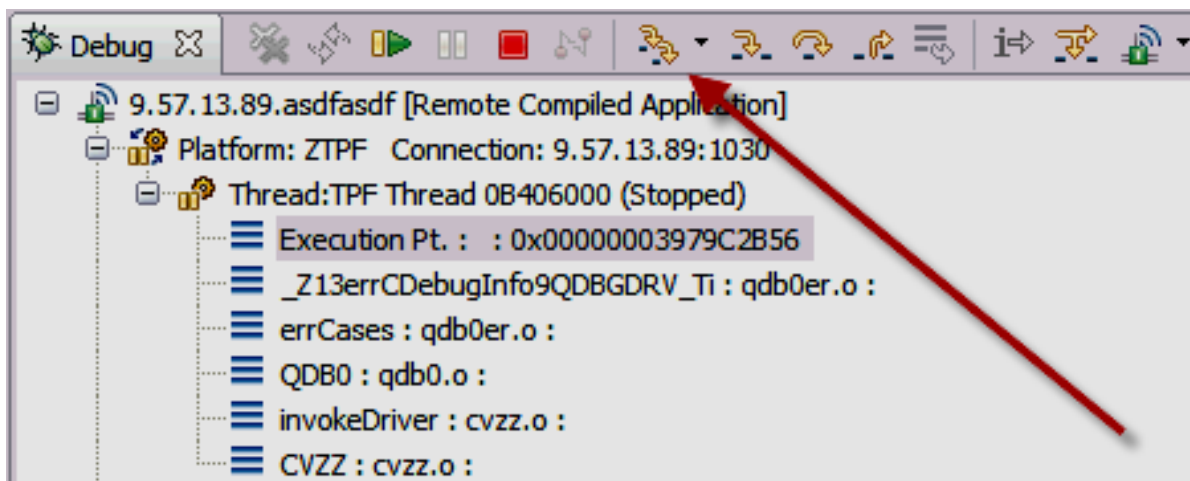
Address	119F8000
Size (user)	7D8
Size (real)	1518
Name	
Corrupted	No
State	Free
Heapcheck	No
ECB SVA	F04E000
Thread id	0
Allocating Program	
Address	409B3637E
Module	QDB0JB
Object	qdb0.cpp
Function	QDB0
Freeing Program	
Address	40ABFBB1C
Module	CPP1
Object	del_op.cc
Function	_ZdlPv

## Malloc View

- **The malloc view provides corruption detection if the corrupt column is visible in any pane. If corruption is being detected, the corrupt blocks will always show in the changed pane.**
- **The malloc view can refresh automatically on each step or set to only refresh when the refresh button is pressed.**
- **The user can also do actions like “go to address” to view the malloc block in the memory view.**
- **Columns can be rearranged, sorted, and hidden.**
- **Names for named malloc entries can also be shown and sorted.**

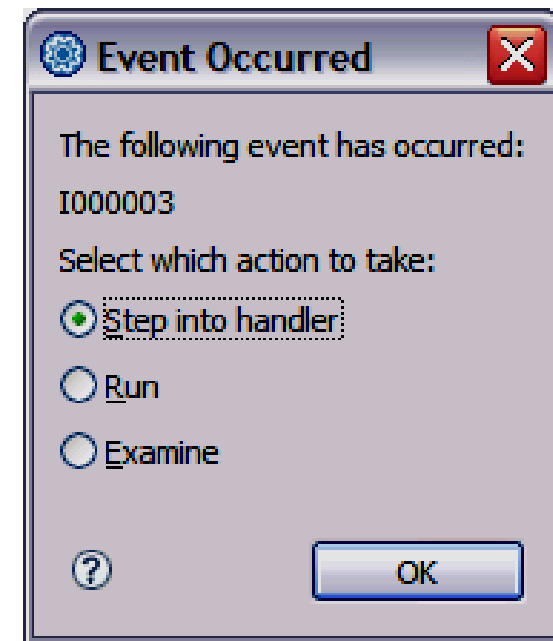
## Auto-Stepping (Trace Run Slow)

- Click the “Animated Step Into” button to set the debugger automatically doing a step into at the specified time interval.
- Click the “Animated Step Into” or another execute (Resume, step into, etc) button to deactivate this feature.
- From the drop down to the right of the “Animated Step Into” button, you can modify the preferences.



## System Error Retry

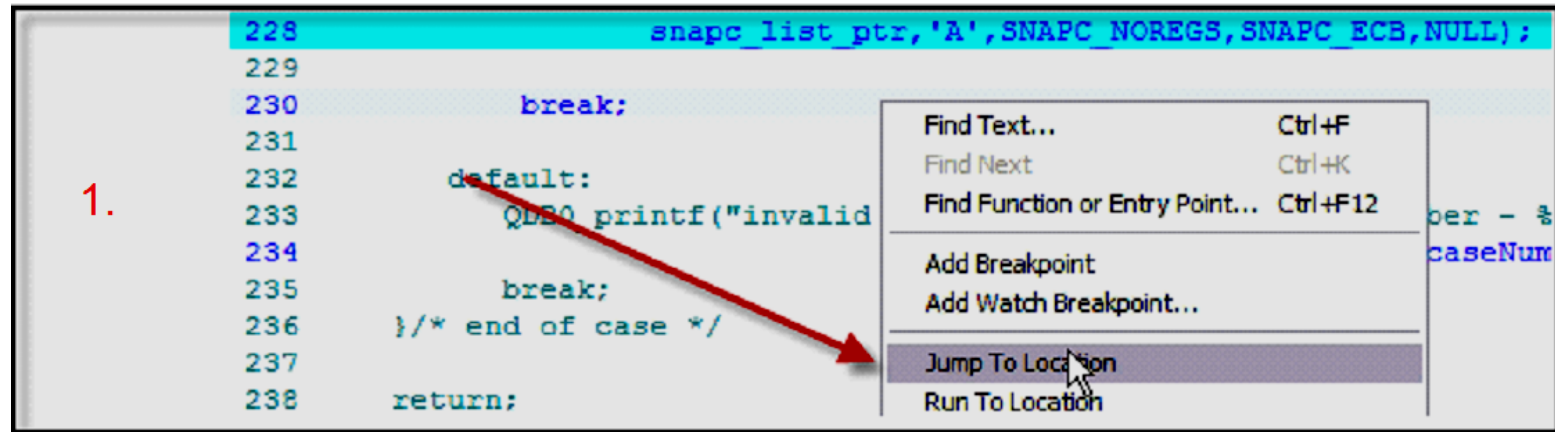
- **Allows you to avoid taking a system error while using the debugger. The debugger shows the application stopped at system error two different ways**
  - 1. Register by System Error starts the debugger at the location of the system error.**
  - 2. While using the debugger, a system error occurs and the user is presented with the “Event Occurred” pop up. In this case, the user must choose Examine in order to use System Error Retry.**





# System Error Retry

- **This feature allows you to avoid the system error in couple different ways.**
  1. Use Jump to location to jump over (bypass) a line causing an error.



The screenshot shows a code editor window with a context menu open. The code is as follows:

```
228     snapc_list_ptr, 'A', SNAPC NOREGS, SNAPC ECB, NULL);
229
230     break;
231
232     default:
233         QD80 printf("invalid
234
235         break;
236     }/* end of case */
237
238     return;
```

The context menu is open over the code, and the 'Jump To Location' option is selected. A red arrow points from the '1.' in the text to the 'Jump To Location' option in the menu.

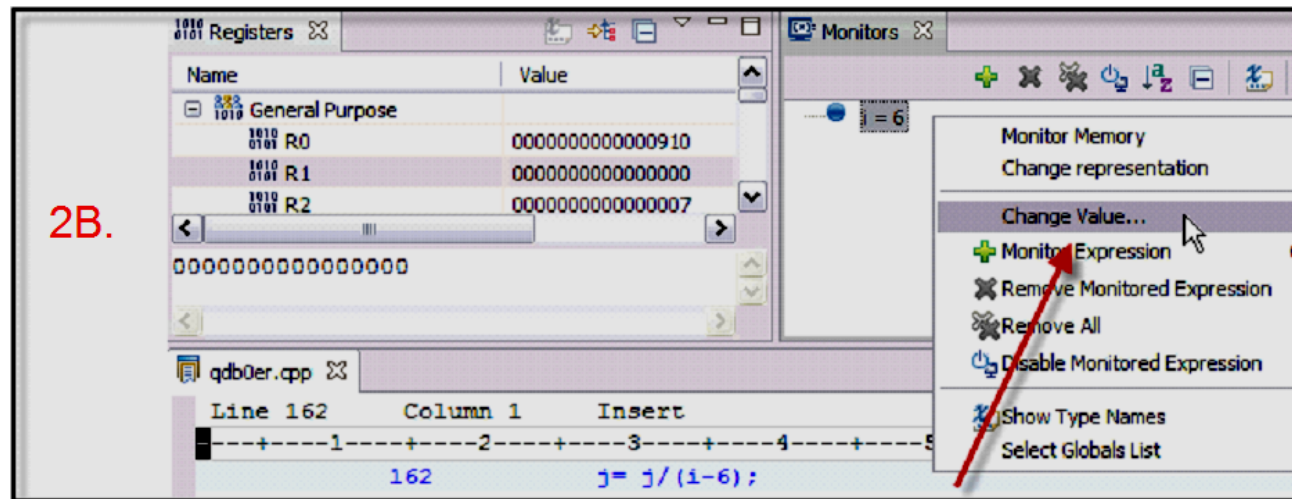
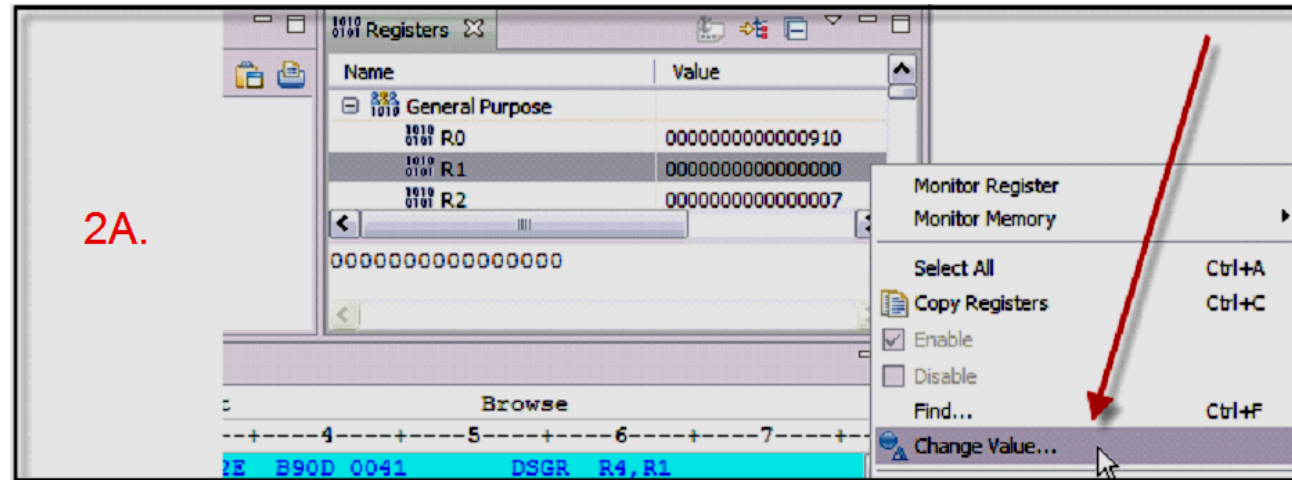
Find Text...	Ctrl+F
Find Next	Ctrl+K
Find Function or Entry Point...	Ctrl+F12
Add Breakpoint	
Add Watch Breakpoint...	
<b>Jump To Location</b>	
Run To Location	

# System Error Retry

## 2. Modify the registers, variables, or memory that is causing the error.

2A. If debugging assembler re-execute the instruction.

2B. If debugging C/C++ use jump to location and re-execute the line.



## Trace Log Enhancement

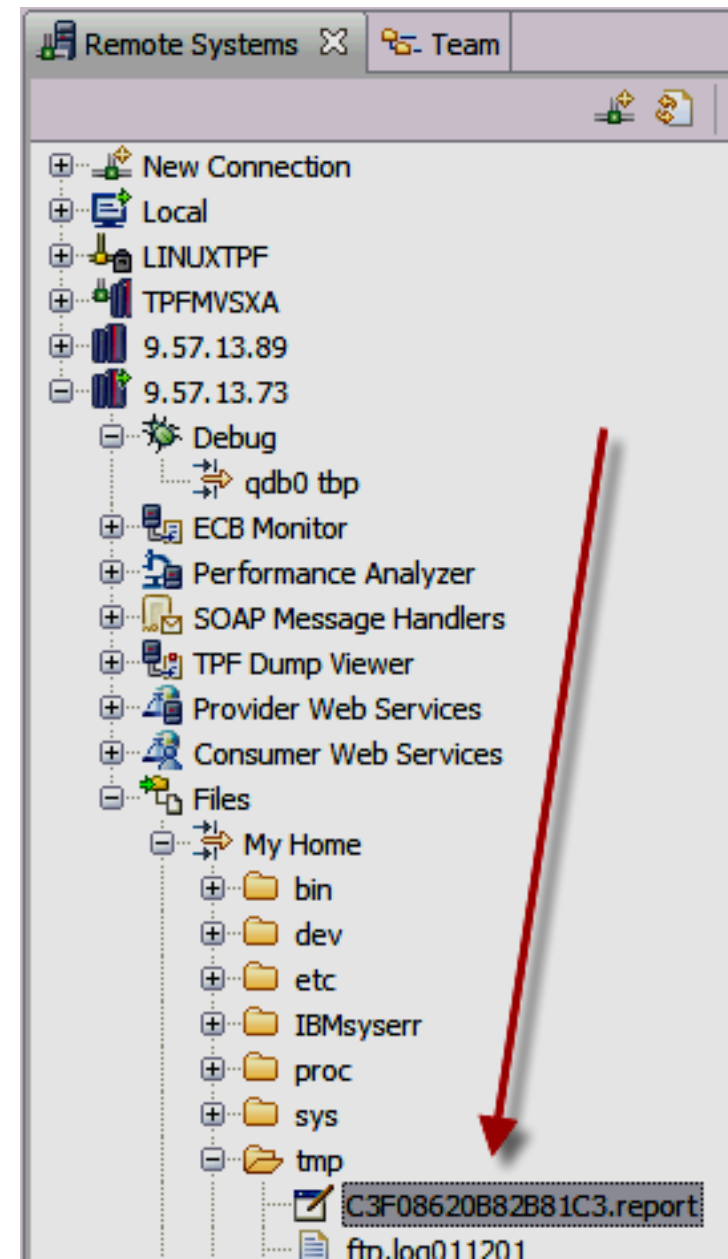
- **Currently, the TRLOG debugger command that is entered through the debug console can only produce a binary format trace log file on the TPF file system. This file must then be post processed offline on Linux.**
- **A new TRLOG parameter has been provided to produce the trace log file in text format with the extension .report such that post processing is not required.**

TRLOG PROC-/directory

- **The .report files can then be opened in LPEX through the TPF Files Subsystem. LPEX provides advanced searching mechanisms.**

# Trace Log Enhancement

- **The Files subsystem is essentially a GUI FTP client. Double clicking the file will open the file in LPEX.**



## Trace Log Enhancement

- Execute the desired searches through LPEX (regular expressions are supported, the regular expression below locates all ENTER and BACKC macro calls for the packages named with QD\* and CX\*)

The screenshot shows the LPEX search interface for a report named 'C3F08620B82B81C3.report'. The main display area shows a list of macro calls with columns for Line, Column, and Insert. The search filter '(ENT.C|BACKC). \*P-(QD|CX)' is entered in the Find field, and a red arrow points to it. The search options are set to 'Regular expression' and 'Wrap'.

Line	Column	Insert
64	1	272 QDBA ENTNC P-QDBB C3F08634 F9887E20
64	1	29E QDB6 ENTNC P-QDB7 C3F08634 FA88E000
31	1	262 QDB7 ENTNC P-QDBC C3F08634 FA88F180
31	1	1AE QDBC ENTNC P-QDB8 C3F08634 FB86CE80
64	1	7CA CPS0 ENTRC P-CXXC C3F08634 FB8719C0

Find: (ENT.C|BACKC). \*P-(QD|CX)

Replace:

Case sensitive
  Whole word
  Regular expression
  Wrap
  Select found text

Restrict search to selection
  Restrict search to columns
 Start column: 1 End column: 80

## Diagnostic Enhancements

1. Registration now includes the workstation name. If the specified IP address fails, DNS will be queried for an IP address to use. However, VPN clients that generate workstation names will still fail to connect. If \* is entered for the workstation name or IP address, the workstation name and IP address will automatically be detected.

**Debug Registration Session**

Workstation Information

Workstation name  Workstation TCP/IP address

TPF Terminal

Terminal name

LNIATA  IP Address  LU Name

## Diagnostic Enhancements

- 2. Re-registering a debug session with a different IP address will now replace any existing registration entry for the same debug session based upon the matching workstation name.**
- 3. If a registration entry is made and other registration entries exist on TPF with the same workstation name but having different workstation IP addresses, the existing registration entries will be updated with the newest IP address.**
- 4. User's can set the connection timeout which is now set to a default of 3 seconds.**
- 5. The originating terminal is now copied to EBROUT of the debugger ECB such that the debugger will issue a WTOPC to the originating terminal for any COMMS errors.**
- 6. The GLUE block and workstation name are now included in dumps such that operators/administrators can follow up with the developer.**

## Debugger with Heap Check mode

- **TPF recommends that your z/TPF test systems are run with Heap Check mode on.**
- **Previously, it was not recommended to have Heap Check mode on when using the Debugger.**
- **This recommendation has now been reversed and the Debugger can be run with Heap Check mode on without any affect on the debugger (Heap Check mode is always turned off for the Debugger regardless of the system setting).**



## CDBPUX User Exit

- On TPF 4.1, the Debug listener would never be running in a production environment which prevented debugger sessions from being started in a production environment.
- On z/TPF, Web Services requires the Debug listener to be running in a production environment.
- The Debugger registration code has been updated to turn on the system hooks when a debugger registration entry is created instead of when the debug listener is started.
- The CDBPUX User Exit provides you the flexibility to allow or prevent registration traffic of your choosing on a given system (for example you could allow dump viewer sessions but prevent all debugger registrations except for a specific IP, User Token or etc).
- The CDBPUX User Exit should be used on production systems to prevent debugger registration (running ECBs should not be debugged/stopped on a production system). However, other debugger features such as the dump viewer and ECB monitor can be used on a production system.
- See `cdbpux.c` for more information.

## Other new features to check out

- **Dump Capture User Exit** – capture and display user specified data in the dump viewer and ECB monitor
- **Event Breakpoints** – Stop at specific C/C++ exceptions with the XCPTRap command or debug all caught exceptions, uncaught exceptions, or system errors.
- **XML Generator for ASM DSECTs** – Automatically generate XML maps with maketpf builds for use in the memory, SW00SR, or other views.

## z/TPF Debugger Deliverable Details

Description	z/TPF APAR	TPF Toolkit Level	TPFUG Requirement
ALASC View DETAC View Watchpoint enhancements Suspend ZASER DBUG policing if no dumps exist	PJ36136 PUT6	V3.4.4	V08009S V08011S V08018S/V08008S Customer Request
ECB Launcher Command TPFMEMfill	PJ36136 PJ36686 PUT6	V3.4.4	V08007S Customer Request
Memory view offset mode	None	V3.4.4	V09101S
Application behavior when debugger dumps	PJ36657 PUT6	None	Customer Request

## z/TPF Debugger Deliverable Details

Description	z/TPF APAR	TPF Toolkit Level	TPFUG Requirement
Register by Function Register by System Error System Error Retry	PJ34615 PUT6	V3.4.0	V08058S
Remote Debug Info ECB Summary View Add Macro Breakpoint	PJ35430 PUT6	V3.4.2	V08061S V08029S
ALLSVC Macro Group	PJ33189 PUT5	None	V08057S
TPFDF Macro Breakpoints DFALL TPFDF Macro Group	PJ35669 PUT6	None	V08055S

## z/TPF Debugger Deliverable Details

Description	z/TPF APAR	TPF Toolkit Level	TPFUG Requirement
Add Module (add breakpoint in any module)	PJ35059 PUT6	V3.2.x	V08062S
Auto-Stepping (trace run slow)	None	V3.4.0	V07009F V08045F
Enhanced Fork Support	PJ34894 PUT6	None	V08030S
Malloc View	PJ36059 PUT6	V3.4.3	V08036F V08031S
Register by User Define (transaction trapping)			V07008F V08001S
Trace Log Enhancement			V08008S
Diagnostic Enhancement			V07013F V08002S
Macro Group List			V08015S

## z/TPF Debugger Deliverable Details

<b>Description</b>	<b>z/TPF APAR</b>	<b>TPF Toolkit Level</b>	<b>TPFUG Requirement</b>
Debugger with heap check mode	PJ34800 PUT6	None	
Dump Capture User Exit	PJ34228 PUT5	None	
CDBPUX User Exit	PJ32209 PJ34474 PUT5	None	
Event Breakpoints	PJ32719 PUT5	None	
XML Generator for ASM DSECTs	PJ31440 PUT5	None	

# Trademarks

- **IBM and TPF Toolkit V3.4 are trademarks of International Business Machines Corporation in the United States, other countries, or both.**
- **Linux is a trademark of Linus Torvalds in the United States, other countries, or both.**
- **Other company, product, or service names may be trademarks or service marks of others.**
- **Notes**
- **Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.**
- **All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.**
- **This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.**
- **All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.**
- **Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.**
- **Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.**
- **This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.**