z/TPF V1.1

TPF Users Group - 2010
TPF Debugger Update

Name: Isa Torres

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.

Agenda

- **Debug Server Access Control**
- **TPF Code Coverage**
- **XML Generator for C/C++**
- **Malloc View Filters**
- **Register by SVC**
- **Instruction Detail Pane**
- **Record Hold Table View**
- **Offset into Dx (datalevel)**
- **Summary of other new features**

Debug Server Access Control

- **Provides a control mechanism to allow or restrict different TPF Toolkit processes that run on a z/TPF System. For example:**
  - Debugger registration (granularity is also available by type)
  - Performance analyzer
  - ECB launcher subsystem
  - Among others…
- **Access control rules are managed (altered and displayed) through the ZDBUG ACCESS command**
- **Access control rules for RSE subsystems in the TPF Toolkit take effect immediately. However, they will not terminate any existing debugger sessions or entry control blocks (ECBs). Use ZDBUG CLEAR to clean existing entries.**
- **These rules are saved across-IPL cycles**
- **No code changes necessary**
- **CDBPUX may be used to follow or override access control in your z/TPF system based on the conditions that you define**

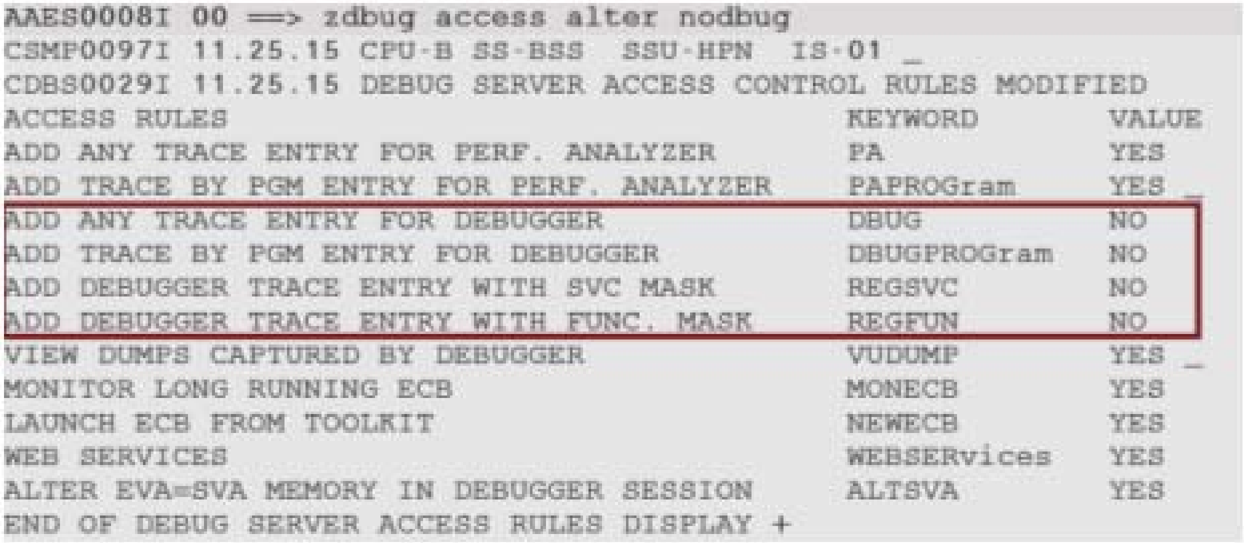
See Appendix A for example with CDBPUX

Debug Server Access Control

- **Example for Access Control**
  - ZDBUG ACCESS ALTER - Use (NO)KEYWORD to alter restrictions

z/TPF System →

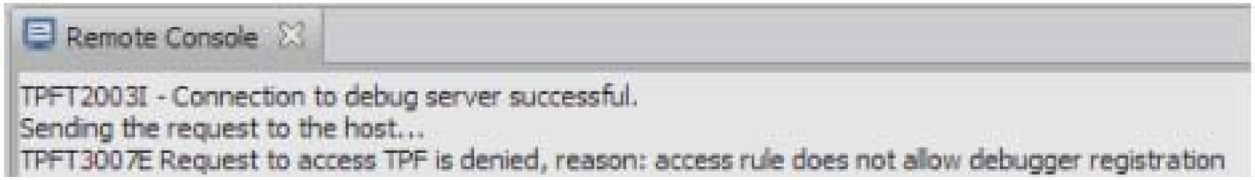
(Do not allow any debug registration request)



```
AAES0008I 00 ==> zdbug access alter nodbug
CSMP0097I 11.25.15 CPU-B SS-BSS  SSU-HPN  IS-01 _
CDBS0029I 11.25.15 DEBUG SERVER ACCESS CONTROL RULES MODIFIED
ACCESS RULES                                  KEYWORD       VALUE
ADD ANY TRACE ENTRY FOR PERF. ANALYZER        PA            YES
ADD TRACE BY PGM ENTRY FOR PERF. ANALYZER     PAPROGram     YES _
ADD ANY TRACE ENTRY FOR DEBUGGER              DBUG          NO
ADD TRACE BY PGM ENTRY FOR DEBUGGER           DBUGPROGram   NO
ADD DEBUGGER TRACE ENTRY WITH SVC MASK        REGSVC        NO
ADD DEBUGGER TRACE ENTRY WITH FUNC. MASK      REGFUN        NO
VIEW DUMPS CAPTURED BY DEBUGGER               VUDUMP        YES _
MONITOR LONG RUNNING ECB                      MONECB        YES
LAUNCH ECB FROM TOOLKIT                       NEWECB        YES
WEB SERVICES                                  WEBSERvices   YES
ALTER EVA=SVA MEMORY IN DEBUGGER SESSION      ALTSVA        YES
END OF DEBUG SERVER ACCESS RULES DISPLAY +
```

TPF Toolkit →

(Debug registration request)



```
TPFT2003I - Connection to debug server successful.
Sending the request to the host...
TPFT3007E Request to access TPF is denied, reason: access rule does not allow debugger registration
```

TPF Code Coverage

- **Designed as a flexible tool for use by Quality Assurance managers, Testers and Developers in Regression, Function, and LCUT test environments.**
- **Intended to communicate the percentage of code executed at different levels to determine if testing coverage is adequate.**
- **Not intended to show the path of execution of an application.**
- **This project is expected to ship in two phases:**
  - Size Analysis – To be available in the near future
  - Source Analysis – Planned for delivery at a later date

TPF Code Coverage

- **Example uses of the code coverage tool:**
  - A QA manager needs to know what modules of an application are not executed when a driver suite is run (hundreds or thousands of modules).
  - A tester needs to ensure that a test plan drives all modules, objects, and functions in an application (tens of modules)
  - A developer needs to ensure that all lines of a new application component have been tested (a few modules at most). Available in Source Analysis.

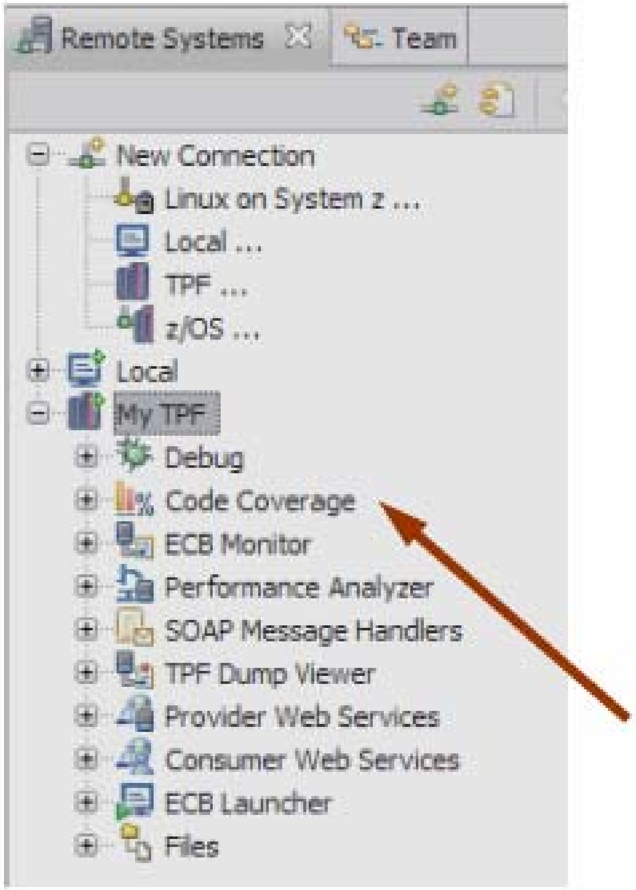
TPF Code Coverage

- **The code coverage tool is broken up into three steps:**
  - Collection: the code coverage tool gathers the number of instructions that have been executed for the module. At this point, after the collection is run, the size percentage is available for viewing at the module level. This collection is done at program level, not at transaction level.
  - Analysis
    - Size Analysis: Breaks the estimated size percentage results down in terms of objects and functions within each module.
  - Viewing: The results of the collection and/or size analysis are presented to the user in the Code Coverage view.

Note: The size percentage is calculated by comparing the number of instructions obtained at collection time with an estimated number of instructions in the module, object, and function.

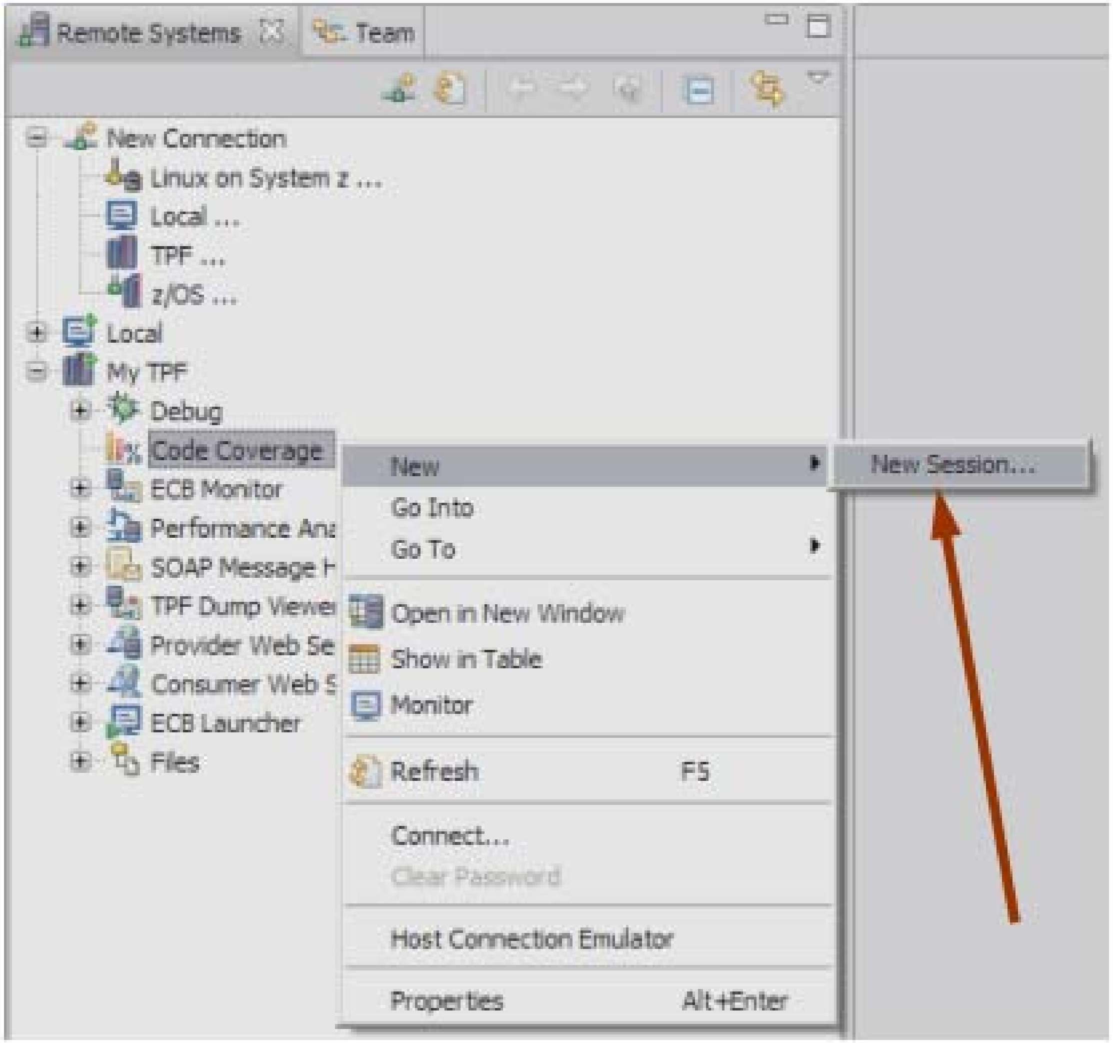
TPF Code Coverage

- **TPF Toolkit Interface**
  - Code Coverage is a new Remote System Explorer (RSE) subsystem under your z/TPF System connection
  - Under this subsystem, you will be able to create and manage Code Coverage registration sessions (performing collection and analysis actions)



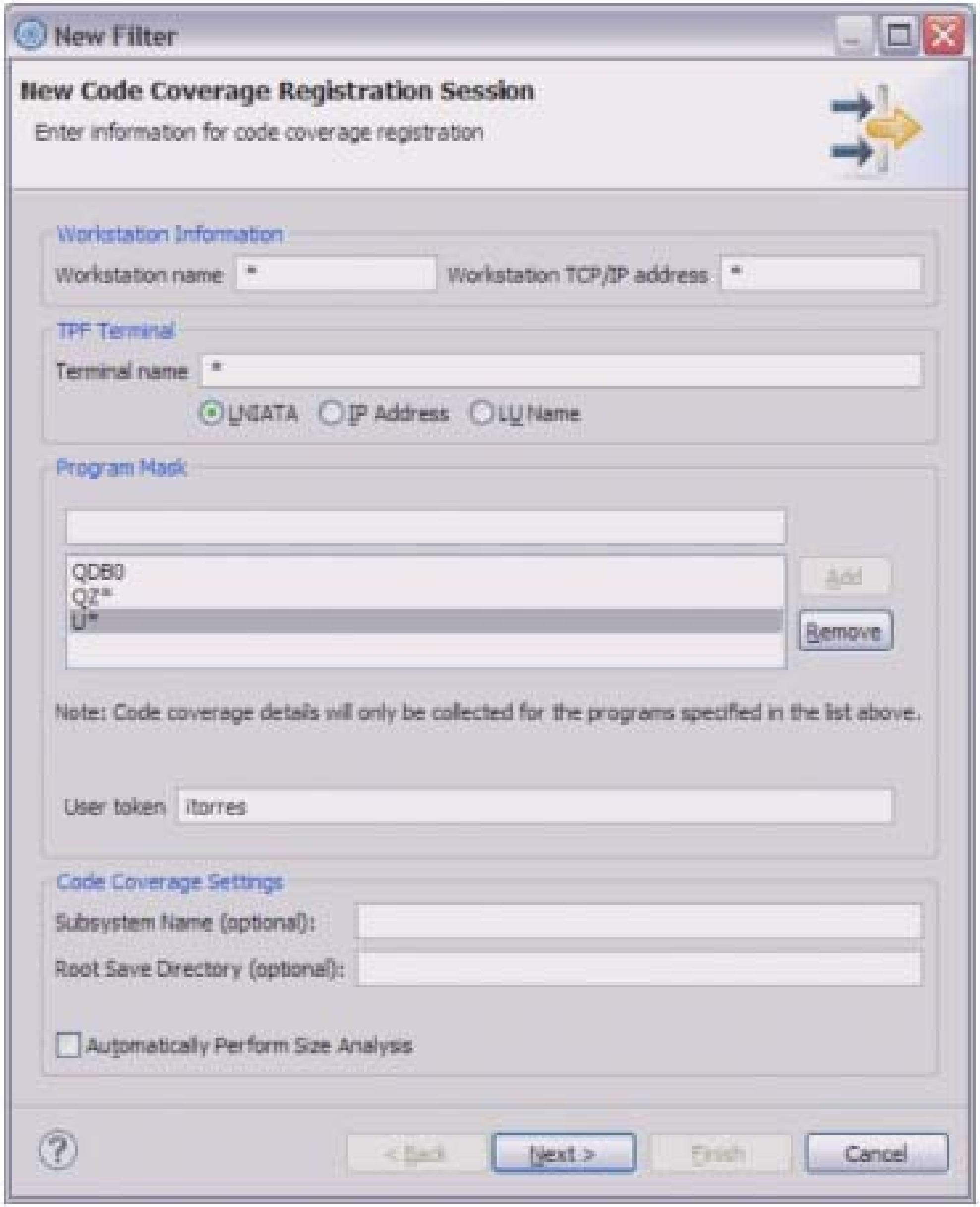
TPF Code Coverage

- **Creating registration sessions is similar to the TPF Debugger, right-click and select New Session**



TPF Code Coverage

- **Code Coverage Collection is done at program level, unlike the debugger which is done at transaction level .**
- **The program mask specifies the list of modules that will have their executed instructions collected.**
- **Specifying a Terminal allows you limit when data for the list of modules is collected**



New Code Coverage Registration Session

Enter information for code coverage registration

Workstation Information

Workstation name: * Workstation TCP/IP address: *

TPF Terminal

Terminal name: *

● LNIATA ○ IP Address ○ LU Name

Program Mask

QDB0
QZ*
U*

Note: Code coverage details will only be collected for the programs specified in the list above.

User token: itorres

Code Coverage Settings

Subsystem Name (optional):

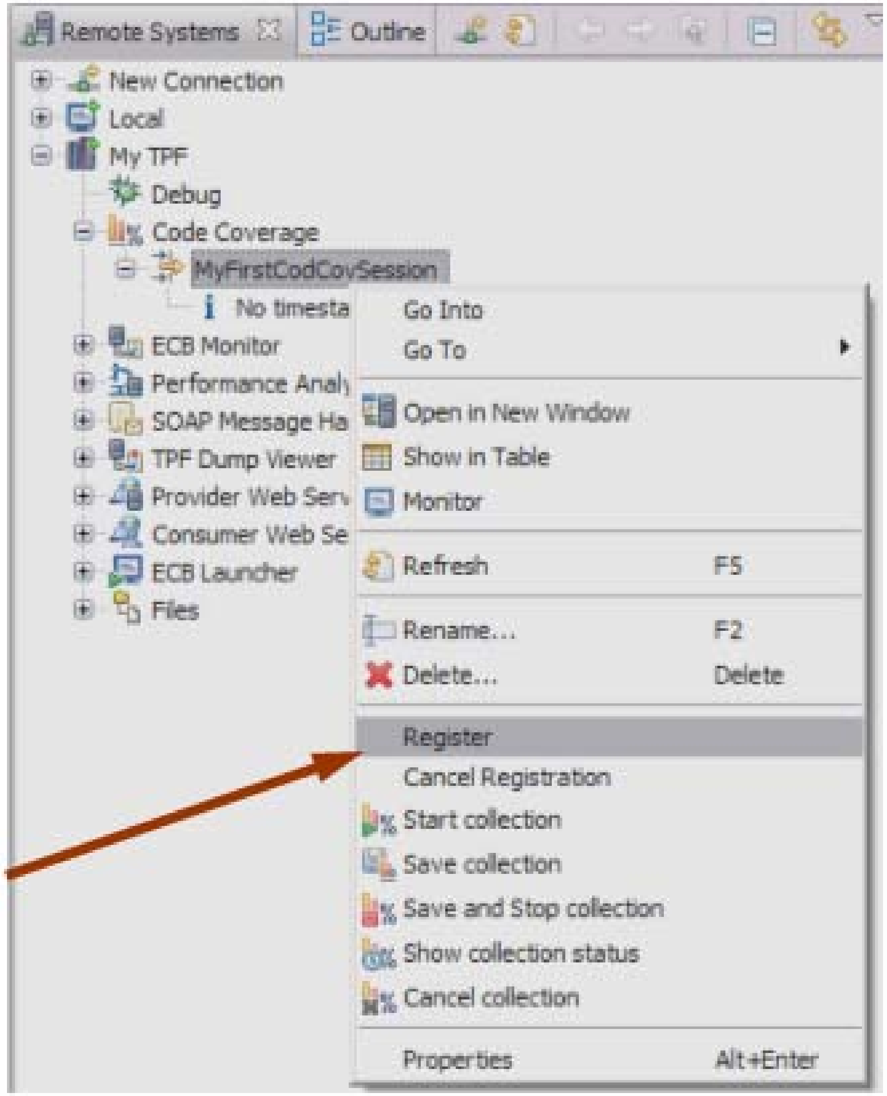
Root Save Directory (optional):

□ Automatically Perform Size Analysis

TPF Code Coverage

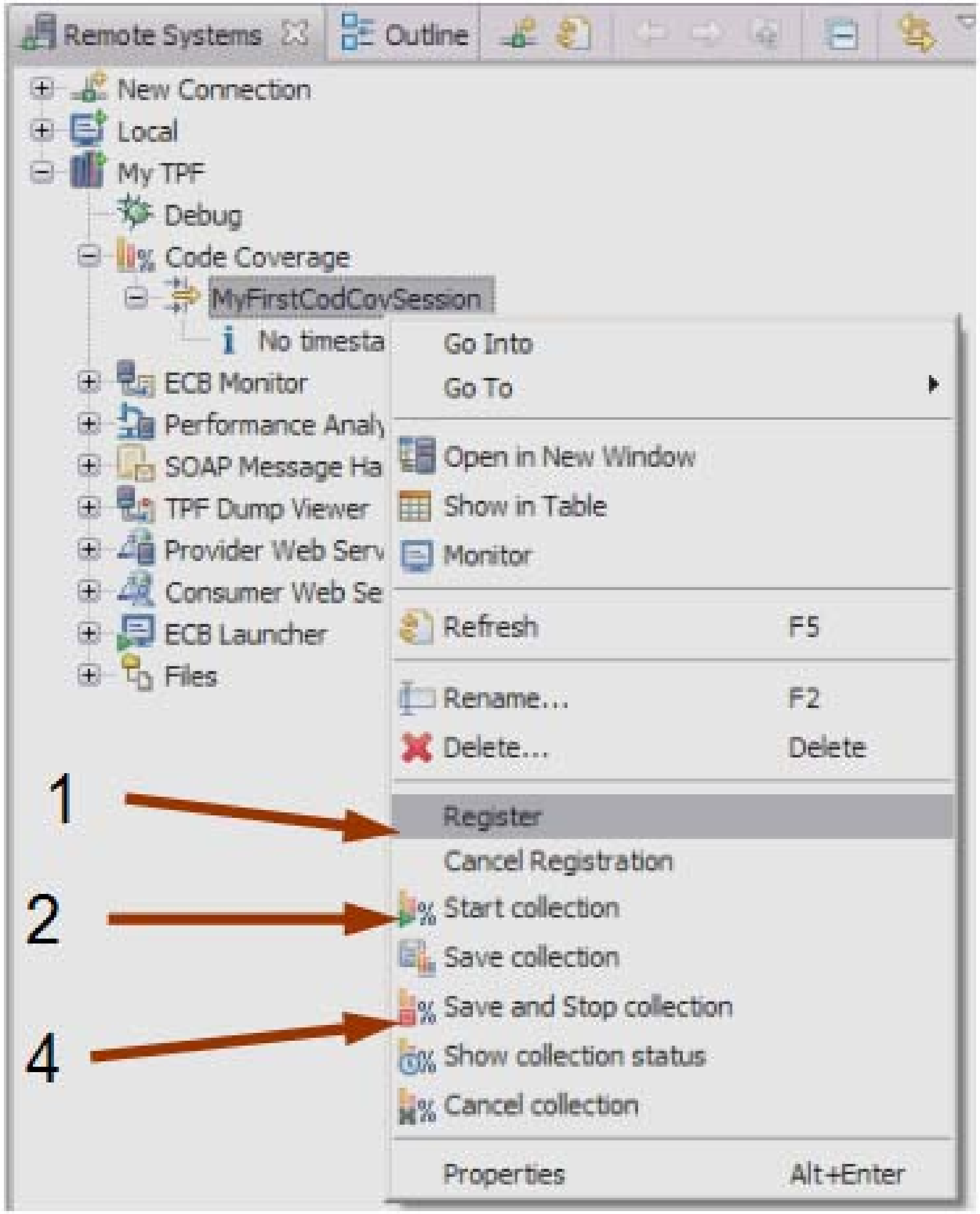
- **Under the Code Coverage subsystem a list of sessions are shown**
- **Right-clicking on a session allows you to perform collection actions on your z/TPF System**

(See Appendix B for action descriptions)



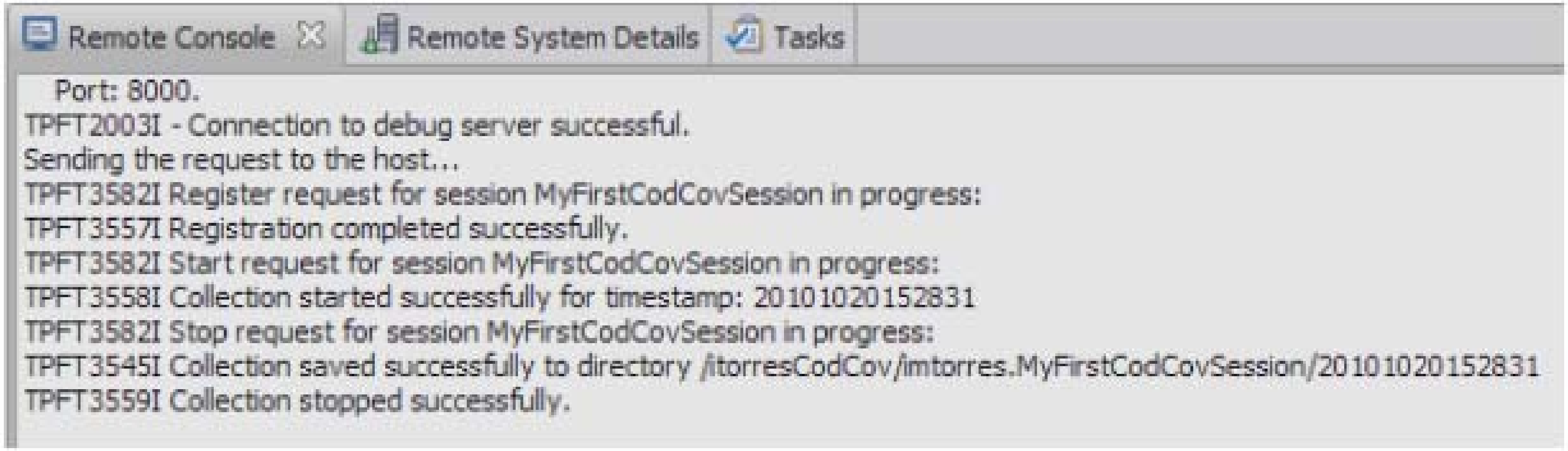
TPF Code Coverage

- **How to perform collection**
  - 1. Register
  - 2. Start collection
  - 3. Drive TPF activity
  - 4. Save/Save and Stop collection



TPF Code Coverage

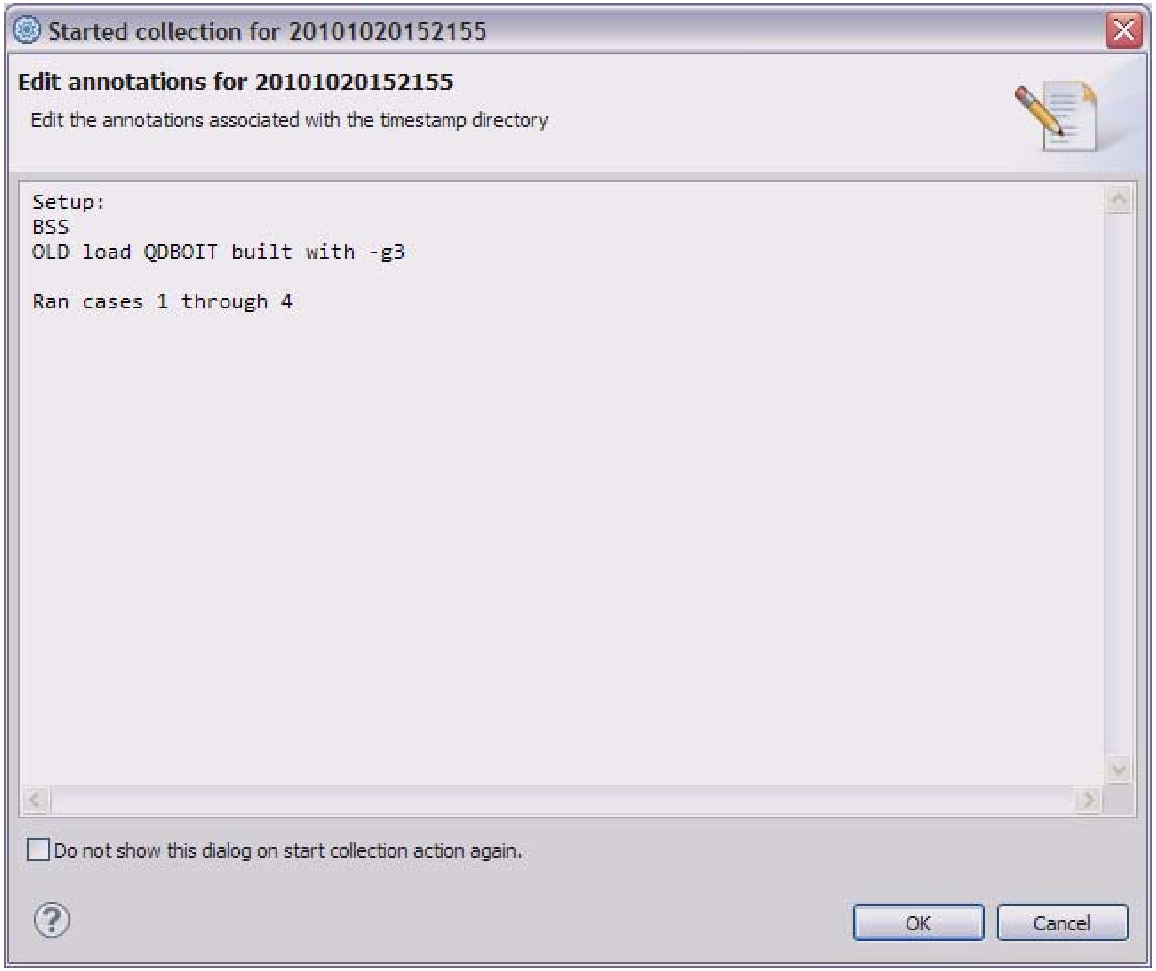
- **Messages regarding session activity are displayed in the Remote Console View in the TPF Toolkit**
- **Messages that require the administrator's attention are displayed in prime CRAS and echoed in the TPF Toolkit**



```
   Port: 8000.
TPFT2003I - Connection to debug server successful.
Sending the request to the host...
TPFT3582I Register request for session MyFirstCodCovSession in progress:
TPFT3557I Registration completed successfully.
TPFT3582I Start request for session MyFirstCodCovSession in progress:
TPFT3558I Collection started successfully for timestamp: 20101020152831
TPFT3582I Stop request for session MyFirstCodCovSession in progress:
TPFT3545I Collection saved successfully to directory /torresCodCov/imtorres.MyFirstCodCovSession/20101020152831
TPFT3559I Collection stopped successfully.
```

TPF Code Coverage

- **The TPF Toolkit provides you with the ability to edit the annotations file associated with collection data.**
- **These annotations can indicate any information or procedure you want to associate with a timestamp directory. This can be used for setup and noting what cases/transactions have been run. It can also be used as a collaboration tool, such as tracking to-do's.**
- **Once you start collection you may be prompted to edit this annotations file.**



Started collection for 20101020152155

Edit annotations for 20101020152155

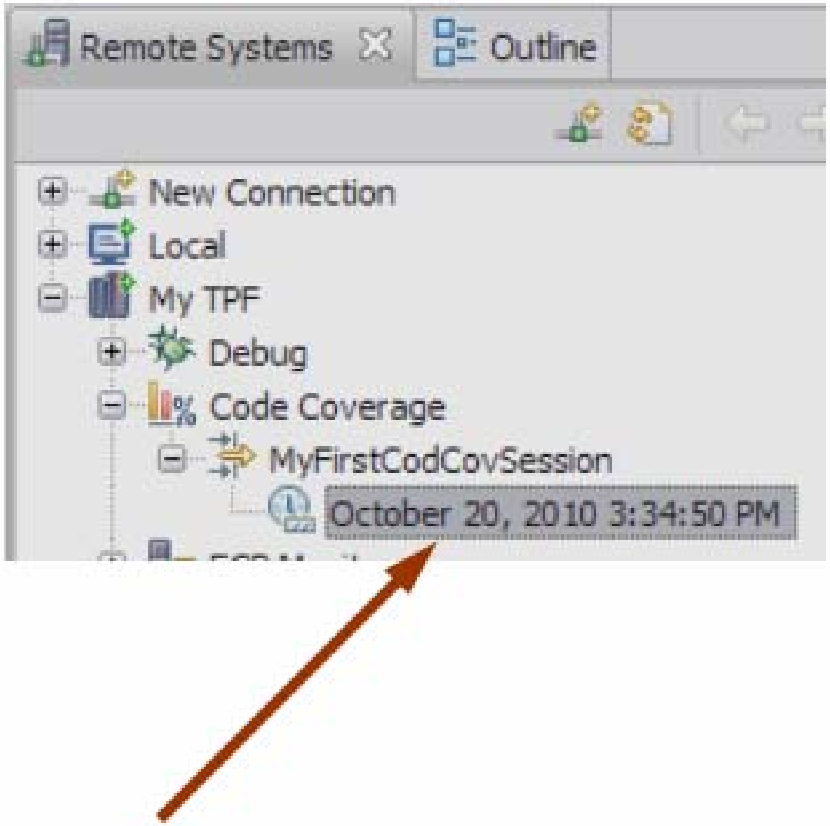
Edit the annotations associated with the timestamp directory

```
Setup:
BSS
OLD load QDBOIT built with -g3

Ran cases 1 through 4
```

□ Do not show this dialog on start collection action again.

TPF Code Coverage

- **Once collected data has been saved, a timestamp directory is created under the registration session**
- **You may perform collection multiple times for a particular registration session**



Remote Systems | Outline

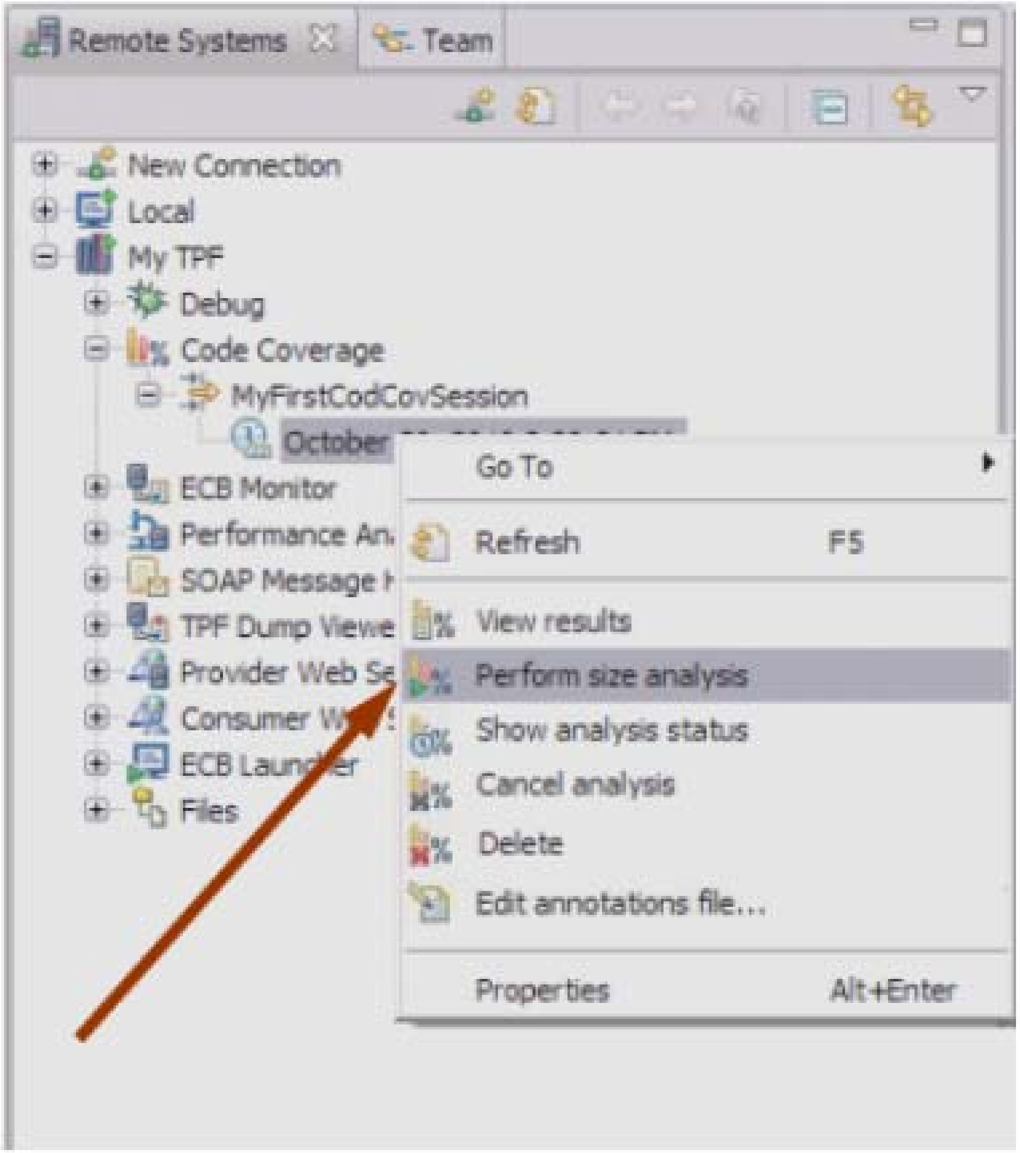
- New Connection
- Local
- My TPF
  - Debug
  - Code Coverage
    - MyFirstCodCovSession
      - October 20, 2010 3:34:50 PM

See Appendix F for more information about the timestamp directory

TPF Code Coverage

- **Once the collection is saved, you may:**
  - View the results at the module level
  - Perform size analysis to generate and display the size percentages at the module, object, and function levels

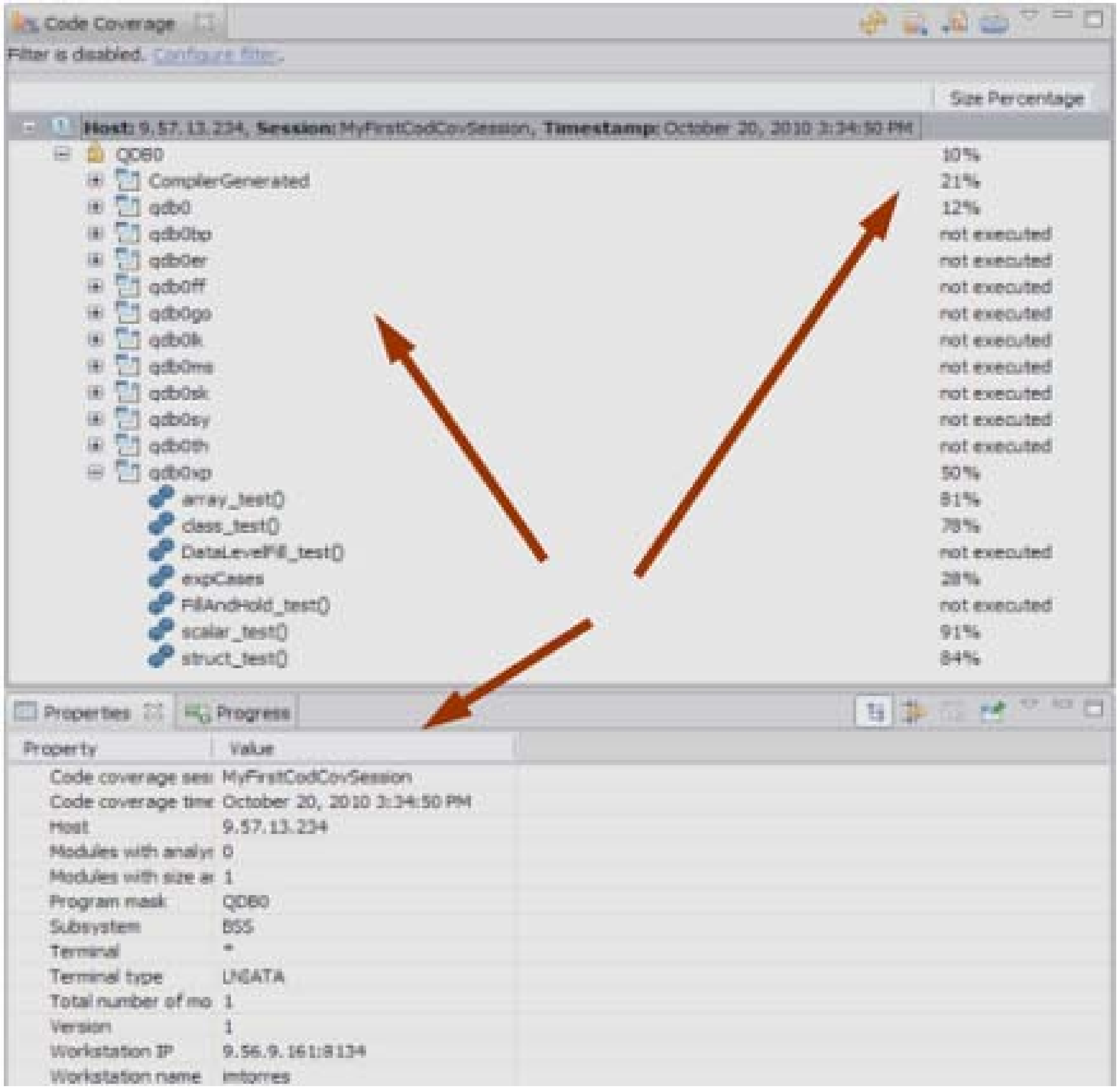
(See Appendix B for action descriptions)



TPF Code Coverage

- **From the Code Coverage view you will see the size percentages for execution**
- **Properties pane provides more information on selected item**
- **These results can be exported to a delimiter-structured text file**

(See Appendix C for more Properties pane examples)



Filter is disabled. Configure filter.

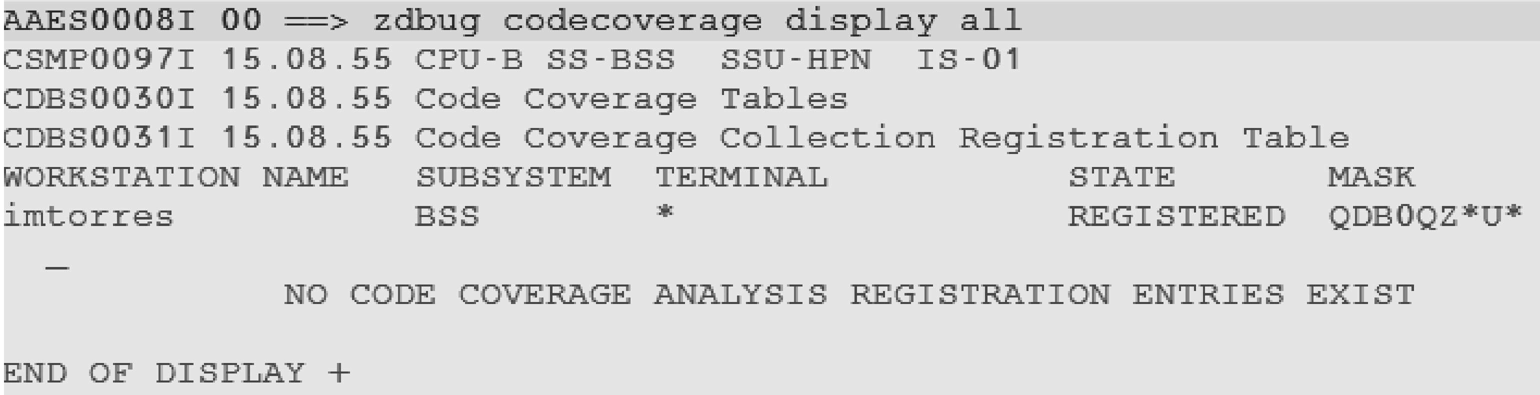
Host: 9.57.13.234, **Session:** MyFirstCodCovSession, **Timestamp:** October 20, 2010 3:34:50 PM

	Size Percentage
QDB0	10%
CompilerGenerated	21%
qdb0	12%
qdb0bp	not executed
qdb0er	not executed
qdb0ff	not executed
qdb0go	not executed
qdb0lk	not executed
qdb0ms	not executed
qdb0sk	not executed
qdb0sy	not executed
qdb0th	not executed
qdb0xp	50%
array_test()	81%
class_test()	78%
DataLevelFill_test()	not executed
expCases	28%
FillAndHold_test()	not executed
scalar_test()	91%
struct_test()	84%

Property	Value
Code coverage ses	MyFirstCodCovSession
Code coverage time	October 20, 2010 3:34:50 PM
Host	9.57.13.234
Modules with analy	0
Modules with size a	1
Program mask	QDB0
Subsystem	BSS
Terminal	*
Terminal type	LNIATA
Total number of mo	1
Version	1
Workstation IP	9.56.9.161:8134
Workstation name	imtorres

TPF Code Coverage

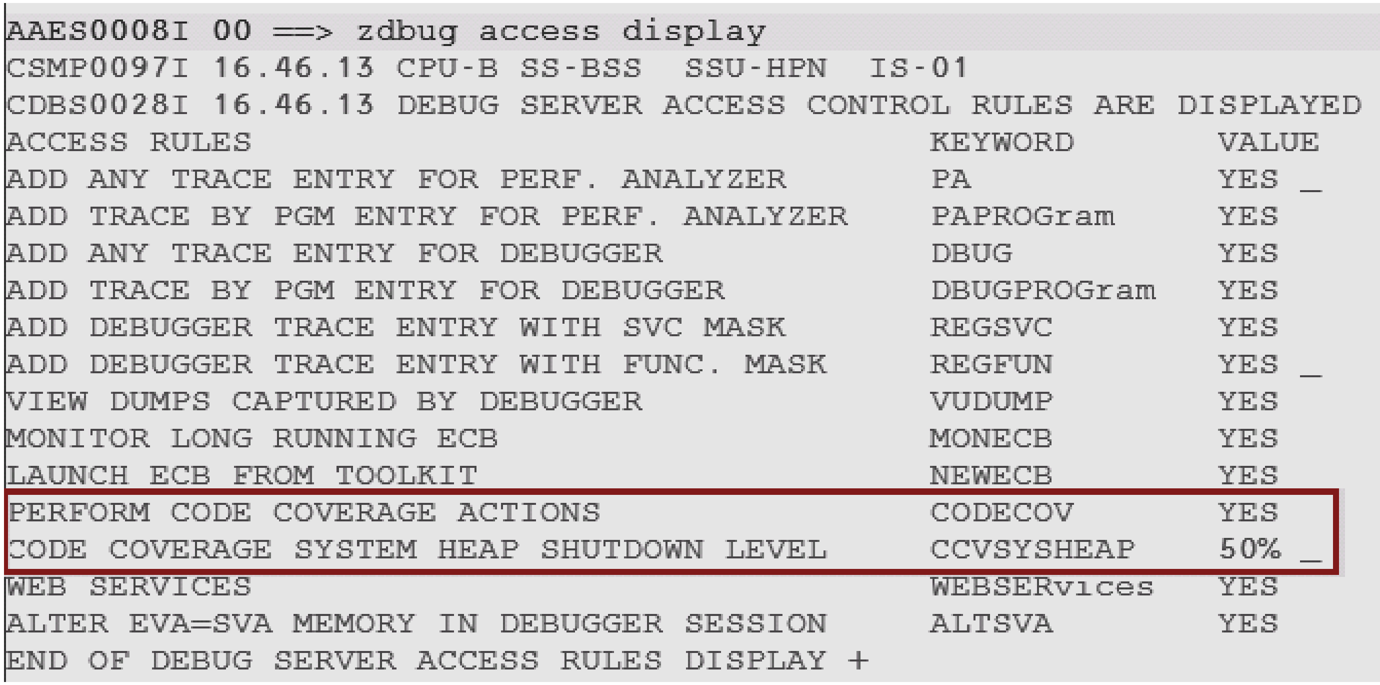
- **ZDBUG/ZDDBG CODecoverage commands are provided to display and manage the Code Coverage registration tables for collection and analysis**



```
AAES0008I 00 ==> zdbug codecoverage display all
CSMP0097I 15.08.55 CPU-B SS-BSS  SSU-HPN  IS-01
CDBS0030I 15.08.55 Code Coverage Tables
CDBS0031I 15.08.55 Code Coverage Collection Registration Table
WORKSTATION NAME   SUBSYSTEM  TERMINAL           STATE       MASK
imtorres           BSS        *                  REGISTERED  QDB0QZ*U*
  _
             NO CODE COVERAGE ANALYSIS REGISTRATION ENTRIES EXIST

END OF DISPLAY +
```

TPF Code Coverage

- **Debug Server Access Control capabilities**
  - TPF Code Coverage also follows Debug Server Access Control rules
  - ZDBUG ALTER (NO)CODECOV restricts/allows code coverage requests to your TPF System
  - ZDBUG ALTER CCVSYSHEAP-xx assigns a new value for the code coverage system heap shutdown level



```
AAES0008I 00 ==> zdbug access display
CSMP0097I 16.46.13 CPU-B SS-BSS  SSU-HPN  IS-01
CDBS0028I 16.46.13 DEBUG SERVER ACCESS CONTROL RULES ARE DISPLAYED
ACCESS RULES                                   KEYWORD        VALUE
ADD ANY TRACE ENTRY FOR PERF. ANALYZER         PA             YES _
ADD TRACE BY PGM ENTRY FOR PERF. ANALYZER      PAPROGram      YES
ADD ANY TRACE ENTRY FOR DEBUGGER               DBUG           YES
ADD TRACE BY PGM ENTRY FOR DEBUGGER            DBUGPROGram    YES
ADD DEBUGGER TRACE ENTRY WITH SVC MASK         REGSVC         YES
ADD DEBUGGER TRACE ENTRY WITH FUNC. MASK       REGFUN         YES _
VIEW DUMPS CAPTURED BY DEBUGGER                VUDUMP         YES
MONITOR LONG RUNNING ECB                       MONECB         YES
LAUNCH ECB FROM TOOLKIT                        NEWECB         YES
PERFORM CODE COVERAGE ACTIONS                  CODECOV        YES
CODE COVERAGE SYSTEM HEAP SHUTDOWN LEVEL       CCVSYSHEAP     50% _
WEB SERVICES                                   WEBSERvices    YES
ALTER EVA=SVA MEMORY IN DEBUGGER SESSION       ALTSVA         YES
END OF DEBUG SERVER ACCESS RULES DISPLAY +
```

TPF Code Coverage

- **Coexistence with TPF Debugger**
  - You are able to debug modules while the code coverage tool is running
  - The code coverage tool will collect the data for the program’s execution during the debug session
  - The registration sessions under Debug and Code Coverage subsystems are independent of each other.

TPF Code Coverage

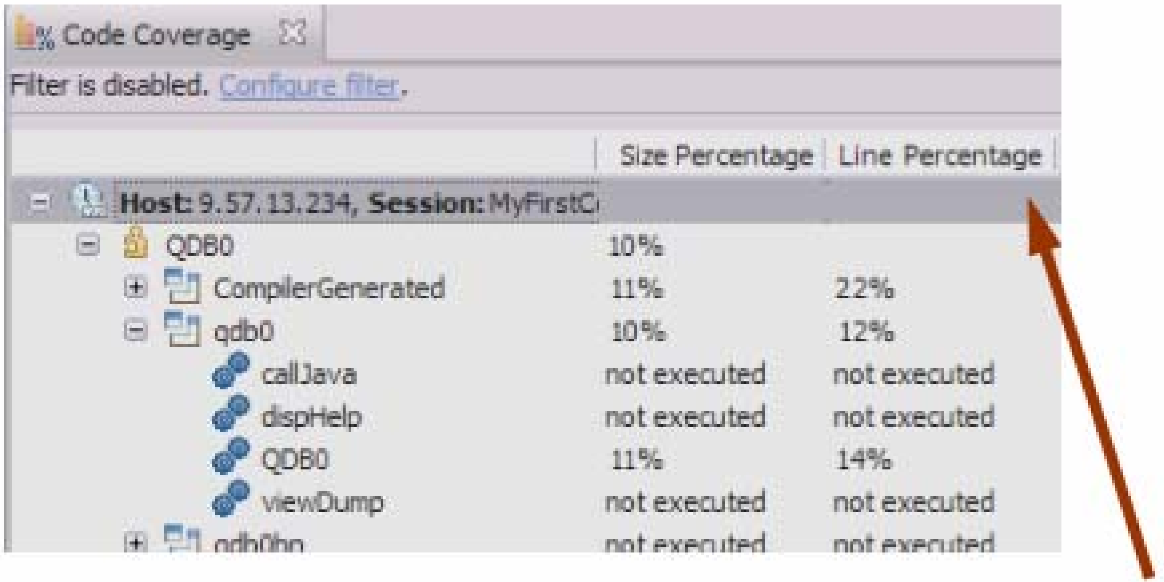
- **System Requirements and Considerations**
  - Your z/TPF System must be at least in 1052 state with pools up and ftp server enabled.
  - Code Coverage runs on the same server on z/TPF as the Debugger. Therefore, the DBUG server must be active on z/TPF to use the Code Coverage tool.
  - A module can only be registered once per system. For example, if program ABC* has been registered, program ABCD cannot be registered in a different session.
  - Selectively activated programs are ignored by code coverage tool at this time.
  - Activating new versions of a program while collection is running may result in unreliable data.

TPF Code Coverage

- **System Requirements and Considerations**
  - Ensure your system has enough system heap available. This is relative to how much data collection is required. The code coverage tool will notify you if more memory is required than there’s available on the system.
  - The code coverage tool is implemented using the Program Event Recording (PER) Facility and as a result it may impact the performance of the system.
  - The code coverage tool should not be used on production systems.

TPF Code Coverage

- **A peek into Phase II (Source Analysis)**
  - Line percentages column in Code Coverage



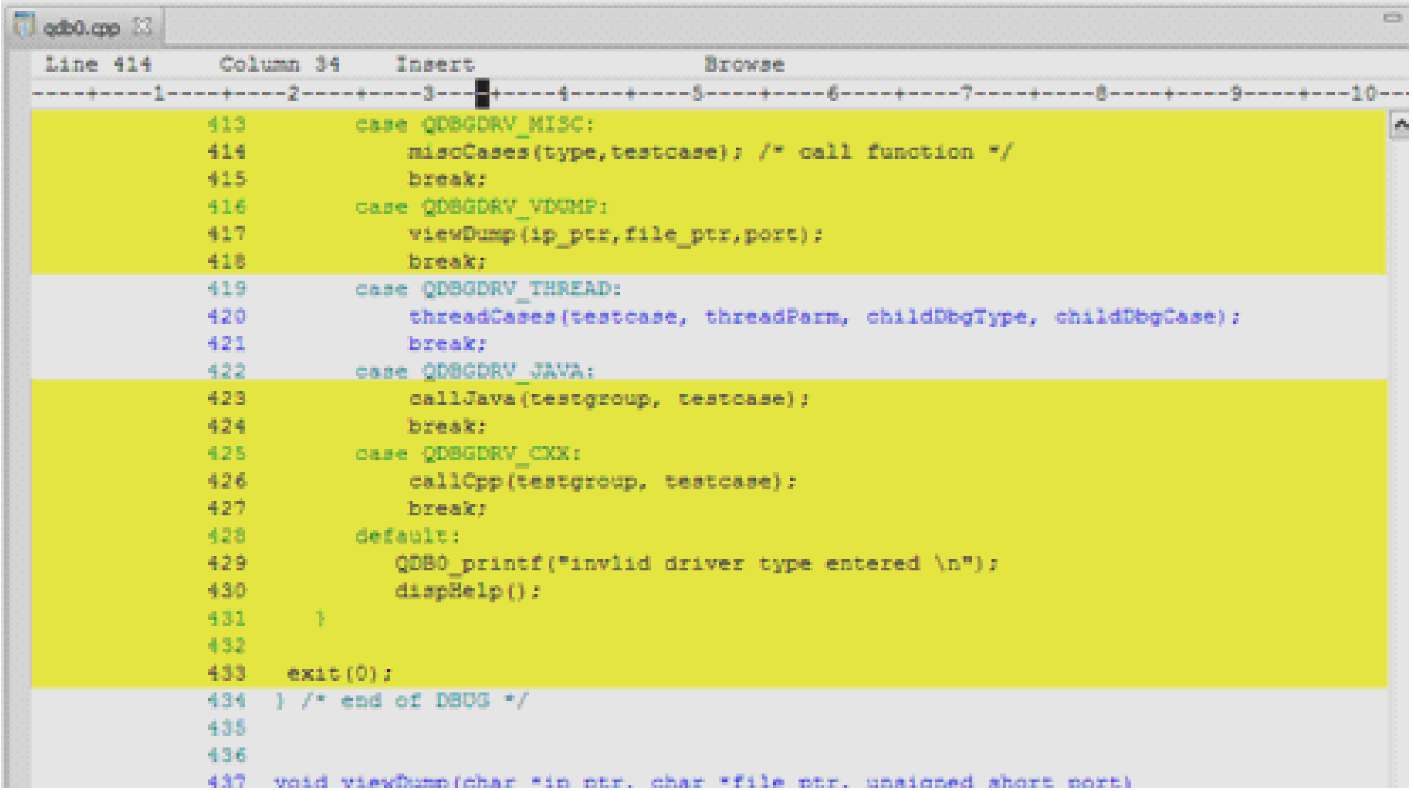
Filter is disabled. Configure filter.

	Size Percentage	Line Percentage
Host: 9.57.13.234, Session: MyFirstC		
QDB0	10%	
CompilerGenerated	11%	22%
qdb0	10%	12%
callJava	not executed	not executed
dispHelp	not executed	not executed
QDB0	11%	14%
viewDump	not executed	not executed
qdb0bo	not executed	not executed

(screen shot generated for illustration purposes only)

TPF Code Coverage

- **A peek into Phase II (Source Analysis)**
  - Source file is shown with executed lines highlighted

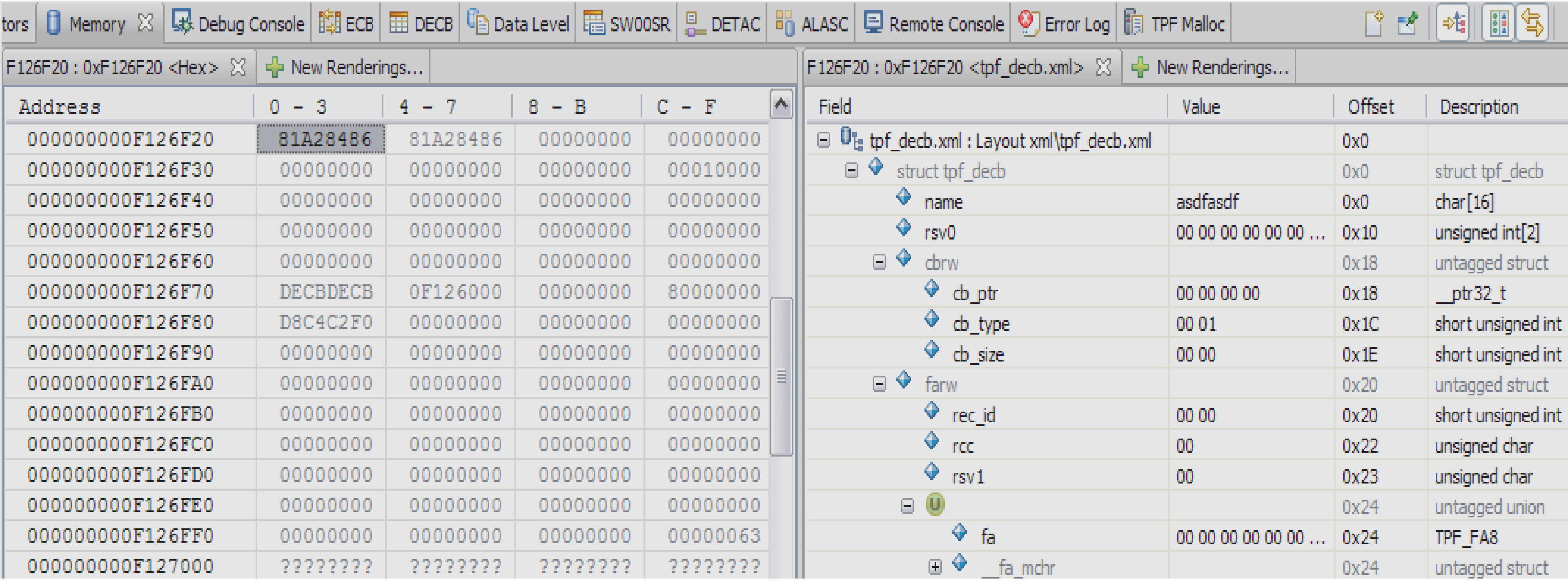


```
Line 414      Column 34     Insert                 Browse
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----10---
      case QDBGDRV_MISC:
          miscCases(type,testcase); /* call function */
          break;
      case QDBGDRV_VDUMP:
          viewDump(ip_ptr,file_ptr,port);
          break;
      case QDBGDRV_THREAD:
          threadCases(testcase, threadParm, childDbgType, childDbgCase);
          break;
      case QDBGDRV_JAVA:
          callJava(testgroup, testcase);
          break;
      case QDBGDRV_CXX:
          callCpp(testgroup, testcase);
          break;
      default:
         QDB0_printf("invlid driver type entered \n");
         dispHelp();
   }

 exit(0);
} /* end of DBUG */


void viewDump(char *ip_ptr, char *file_ptr, unsigned short port)
```

(screen shot generated for illustration purposes only)

XML Generator for C/C++

- **Provides users with the ability to generate XML map files for structures/classes defined in C/C++ objects**
- **These XML map files may be used as renderings in the Memory View to map C/C++ structures to monitored memory**



Address	0 - 3	4 - 7	8 - B	C - F
000000000F126F20	81A28486	81A28486	00000000	00000000
000000000F126F30	00000000	00000000	00000000	00010000
000000000F126F40	00000000	00000000	00000000	00000000
000000000F126F50	00000000	00000000	00000000	00000000
000000000F126F60	00000000	00000000	00000000	00000000
000000000F126F70	DECBDECB	0F126000	00000000	80000000
000000000F126F80	D8C4C2F0	00000000	00000000	00000000
000000000F126F90	00000000	00000000	00000000	00000000
000000000F126FA0	00000000	00000000	00000000	00000000
000000000F126FB0	00000000	00000000	00000000	00000000
000000000F126FC0	00000000	00000000	00000000	00000000
000000000F126FD0	00000000	00000000	00000000	00000000
000000000F126FE0	00000000	00000000	00000000	00000000
000000000F126FF0	00000000	00000000	00000000	00000063
000000000F127000	????????	????????	????????	????????

Field	Value	Offset	Description
tpf_decb.xml : Layout xml\tpf_decb.xml		0x0	
struct tpf_decb		0x0	struct tpf_decb
name	asdfasdf	0x0	char[16]
rsv0	00 00 00 00 00 00 ...	0x10	unsigned int[2]
cbrw		0x18	untagged struct
cb_ptr	00 00 00 00	0x18	__ptr32_t
cb_type	00 01	0x1C	short unsigned int
cb_size	00 00	0x1E	short unsigned int
farw		0x20	untagged struct
rec_id	00 00	0x20	short unsigned int
rcc	00	0x22	unsigned char
rsv1	00	0x23	unsigned char
U		0x24	untagged union
fa	00 00 00 00 00 00 ...	0x24	TPF_FA8
__fa_mchr		0x24	untagged struct

See Appendix D For more information

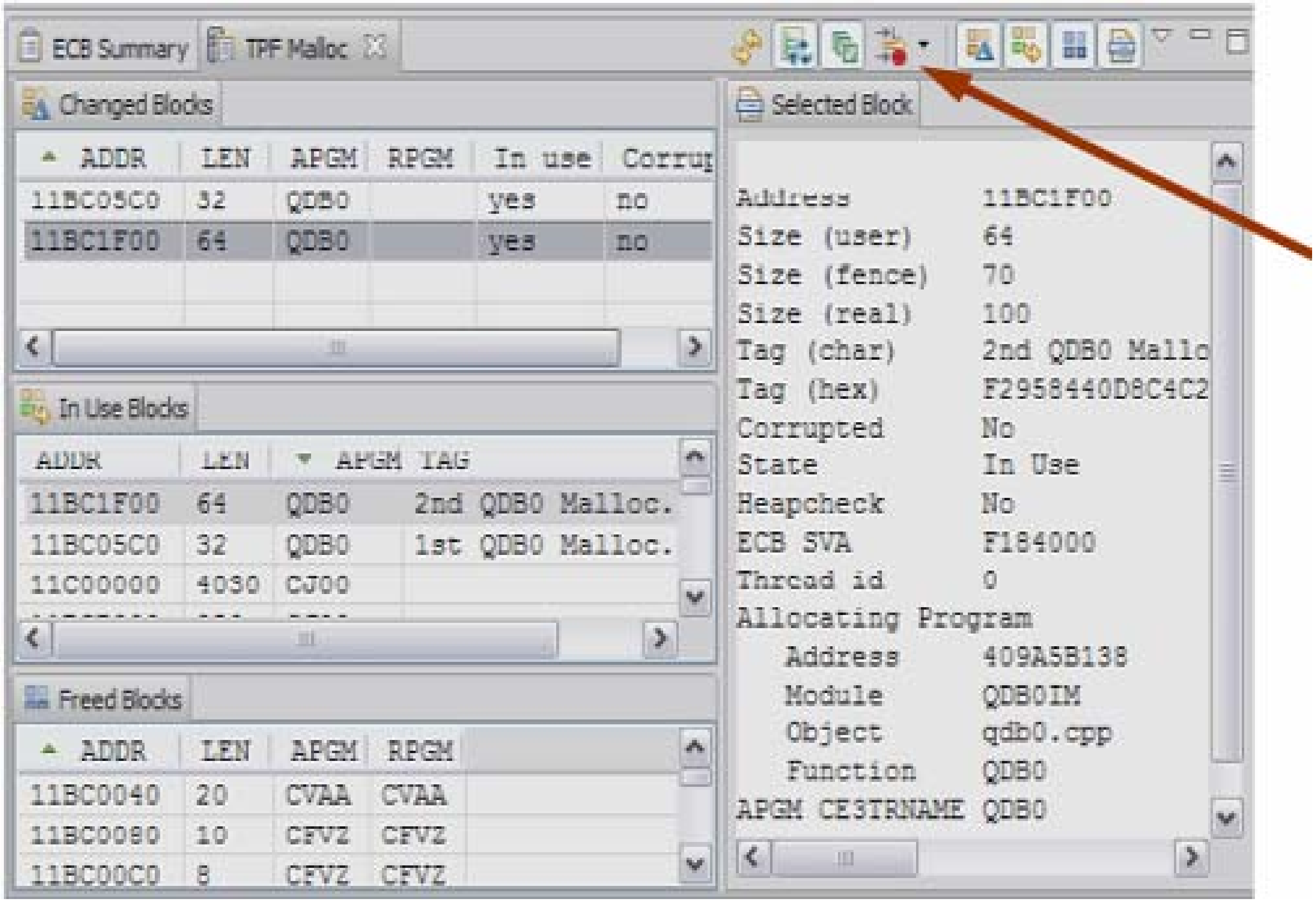
XML Generator for C/C++

- **To automatically generate these XML map files at build time, include the XMLGEN option to CFLAGS_USER and/or CXXFLAGS_USER fields. The files will be located under your project's base/xml directory**
- **You may also manually generate these files at any time by issuing the** `tpfxmlgen` **command from linuxtpf**
- **When the** `tpfxmlgen` **command is manually issued, you may specify a list of DSECTs/Structures/Classes. This will act as a filter list generating only the XML map files you're interested in**

```
itorres@linuxtpf:~/> tpfxmlgen qdb0xp.o -f eb0eb,classa,_structa

Creating eb0eb.xml

Creating classa.xml

Creating _structa.xml

itorres@linuxtpf:~/> ls *.xml

classa.xml  eb0eb.xml  _structa.xml
```

Malloc View Filters

- **Allows the user to filter the malloc entries shown in the TPF Malloc View**



Changed Blocks

ADDR	LEN	APGM	RPGM	In use	Corru
11BC05C0	32	QDB0		yes	no
11BC1F00	64	QDB0		yes	no

In Use Blocks

ADDR	LEN	APGM	TAG
11BC1F00	64	QDB0	2nd QDB0 Malloc.
11BC05C0	32	QDB0	1st QDB0 Malloc.
11C00000	4030	CJ00	

Freed Blocks

ADDR	LEN	APGM	RPGM
11BC0040	20	CVAA	CVAA
11BC0080	10	CFVZ	CFVZ
11BC00C0	8	CFVZ	CFVZ

Selected Block

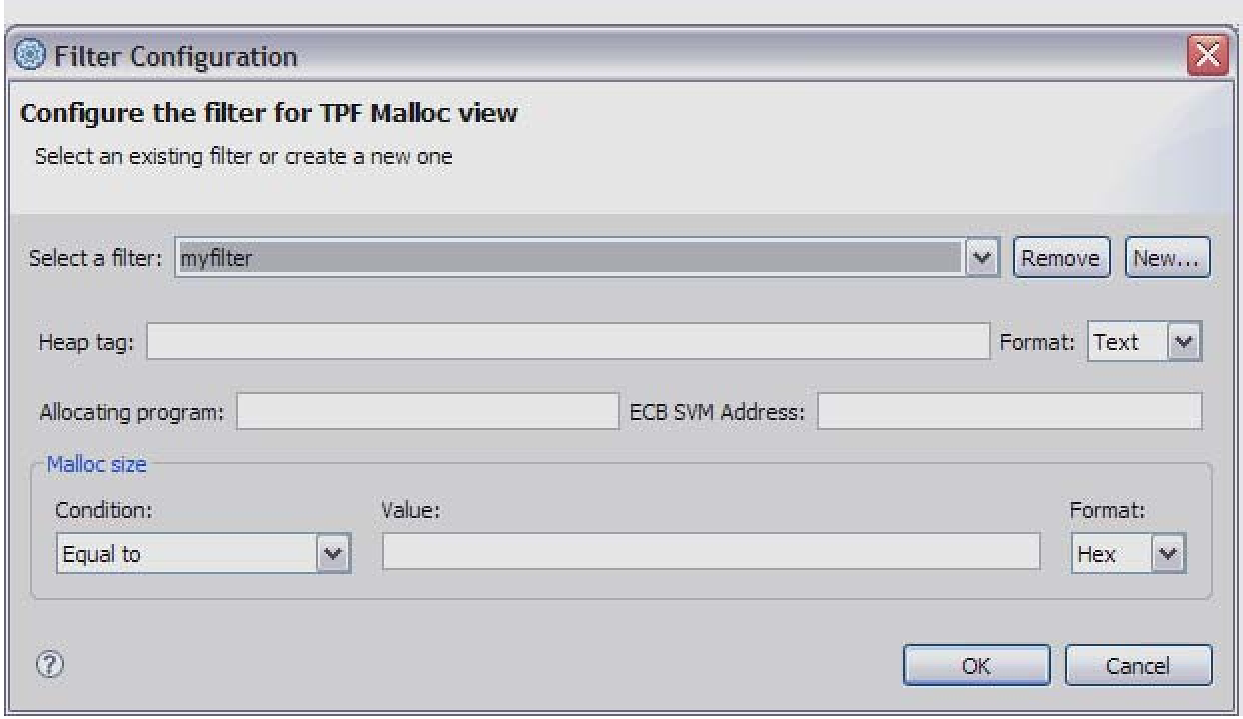
```
Address            11BC1F00
Size (user)        64
Size (fence)       70
Size (real)        100
Tag (char)         2nd QDB0 Mallo
Tag (hex)          F2958440D8C4C2
Corrupted          No
State              In Use
Heapcheck          No
ECB SVA            F184000
Thread id          0
Allocating Program
   Address         409A5B138
   Module          QDB0IM
   Object          qdb0.cpp
   Function        QDB0
APGM CE3TRNAME     QDB0
```

Enable/Disable Malloc View filter

(toggle button)

Malloc View Filters

- **Malloc View filters are created using the following criteria:**
  - Heap tag - heap tag assigned through the tpf_eheap_tag() function or EHEAPC macro. Wild card characters are accepted.
  - Allocating program name – name of program that allocated the malloc block. Wild card characters are accepted.
  - Malloc size - size of each malloc block. Tests: greater than, equal to, less than a given size or within a size range.
  - ECB SVM address - ECB address that performed malloc. Useful for threaded applications.



Filter Configuration

Configure the filter for TPF Malloc view

Select an existing filter or create a new one

Select a filter: myfilter

Heap tag:

Format: Text

Allocating program:

ECB SVM Address:

Malloc size

Condition: Equal to

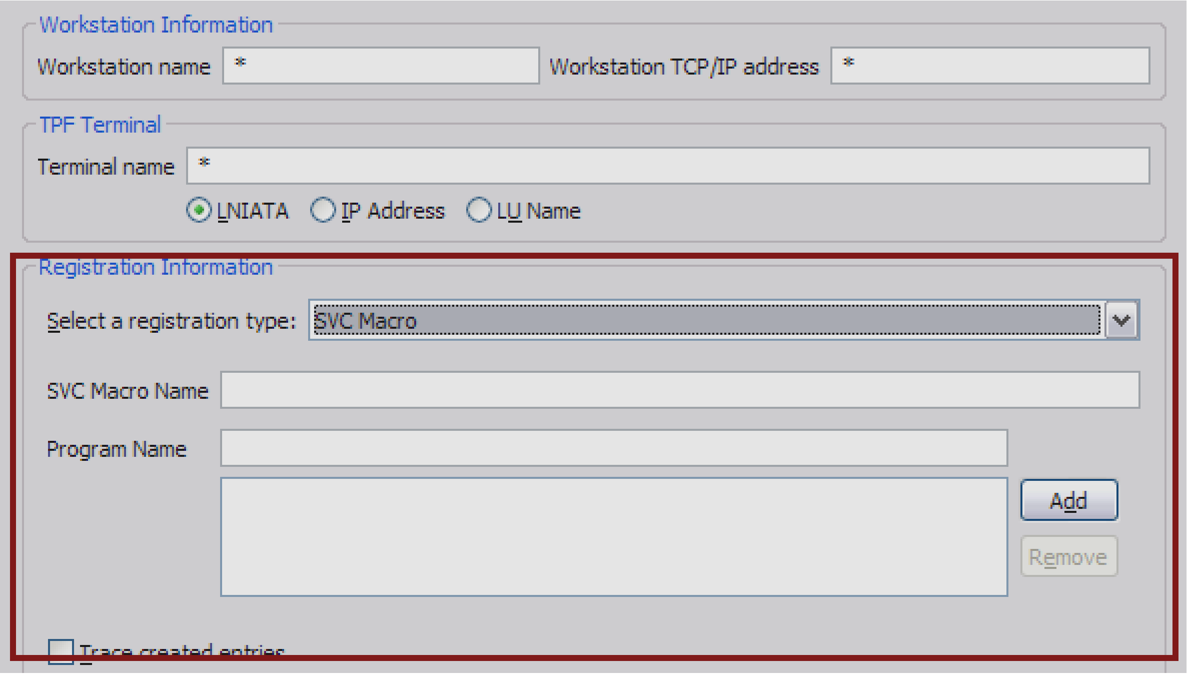
Value:

Format: Hex

(See Appendix G for more information)

Register by SVC

- **Provides the ability to register an SVC macro on which to start a debugger session**
- **The program name is a required parameter and may contain wild card**



Workstation Information

Workstation name: *

Workstation TCP/IP address: *

TPF Terminal

Terminal name: *

● LNIATA ○ IP Address ○ LU Name

Registration Information

Select a registration type: SVC Macro

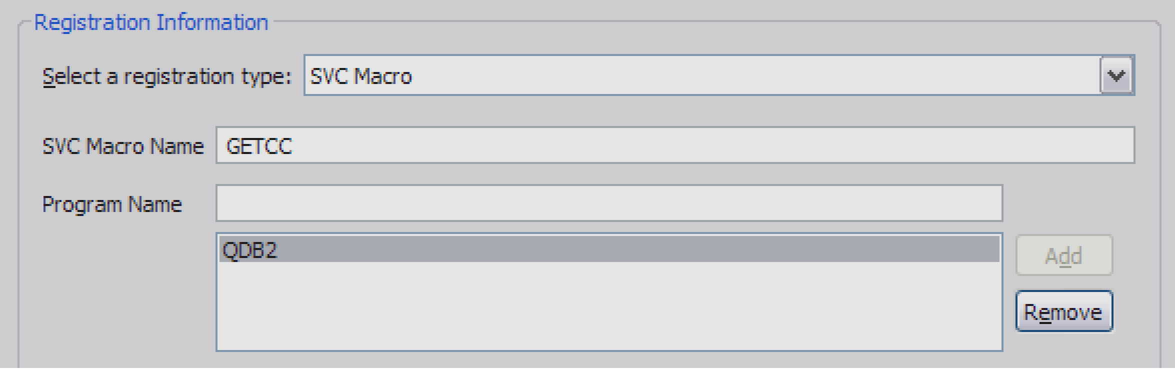
SVC Macro Name:

Program Name:

□ Trace created entries

Register by SVC

- **Example: start a debugger session when GETCC macro is issued from program QDB2**



Registration Information

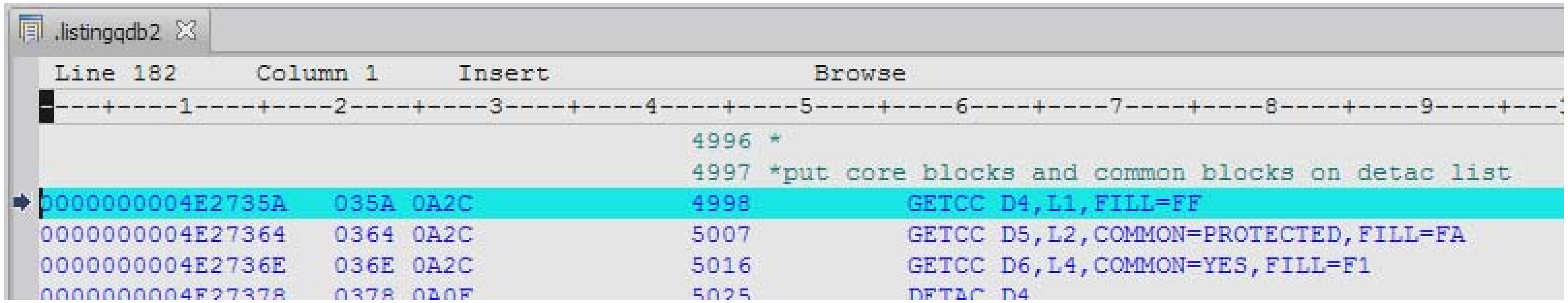
Select a registration type: SVC Macro

SVC Macro Name: GETCC

Program Name:

QDB2

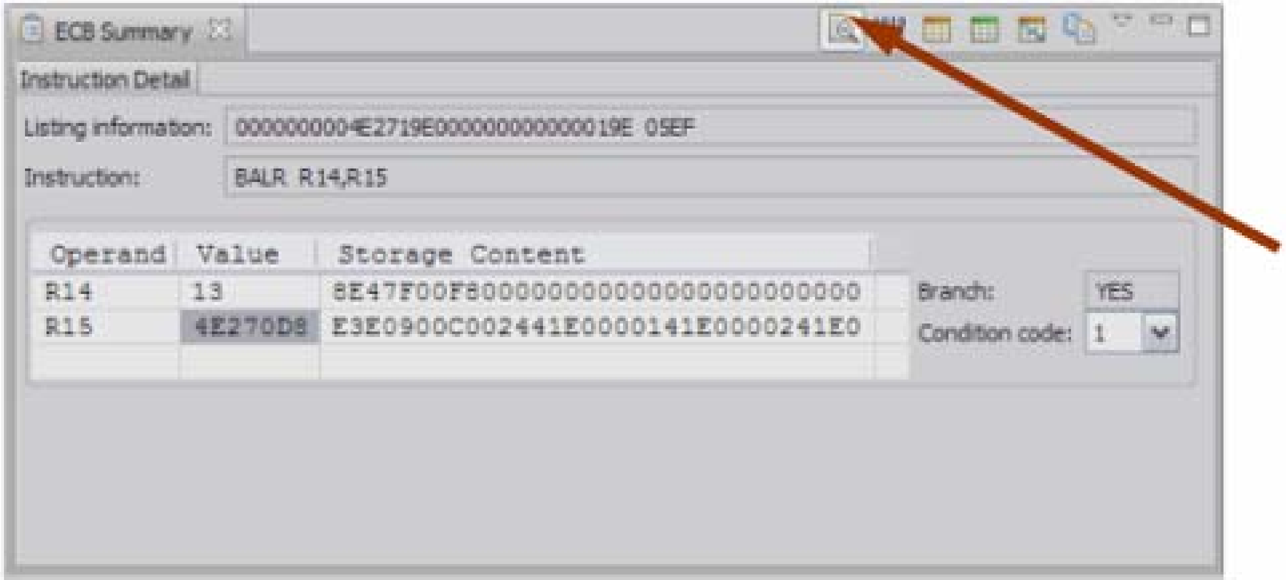
- **Debugger starts at location where GETCC is issued, prior to macro execution**



```
 Line 182     Column 1     Insert                Browse
 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+---
                                        4996 *
                                        4997 *put core blocks and common blocks on detac list
000000004E2735A    035A 0A2C            4998            GETCC D4,L1,FILL=FF
000000004E27364    0364 0A2C            5007            GETCC D5,L2,COMMON=PROTECTED,FILL=FA
000000004E2736E    036E 0A2C            5016            GETCC D6,L4,COMMON=YES,FILL=F1
000000004E27378    0378 0A0E            5025            DETAC D4
```

Instruction Detail Pane

- **Provides a new pane in the ECB Summary View which displays the details of the current instruction**
  - Instruction details from listing
  - Instruction as coded
  - Instruction operands (storage content can be altered by user)
  - Condition Code (can be altered by user)
  - Branch Indicator (YES/NO)
- **This is available in Source and Disassembly View for ASM code.**
- **For C/C++ code, this is only available in Disassembly View.**



Instruction Detail

Listing information: 000000004E2719E000000000000019E 05EF

Instruction: BALR R14,R15

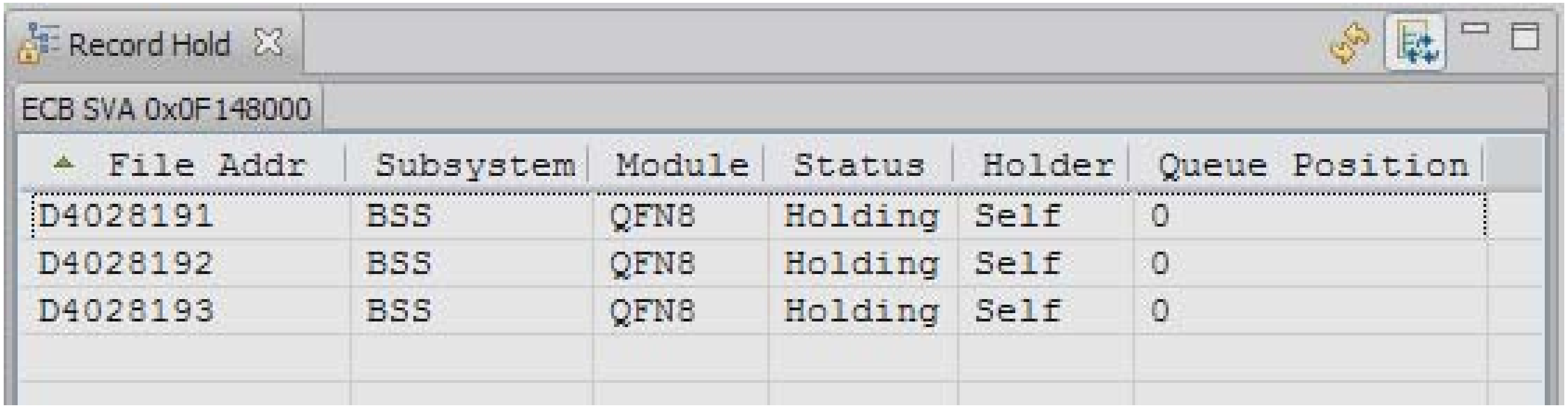
Operand	Value	Storage Content
R14	13	8E47F00F800000000000000000000000
R15	4E270D8	E3E0900C002441E0000141E0000241E0

Branch: YES

Condition code: 1

Record Hold Table View

- **This view displays all records held and waited on by the currently debugged ECB on a thread basis**
- **Unlike ZDRHT, this display focuses only on records that are specific to the current ECB**
- **This view is not available in Dump Viewer or ECB Monitor**



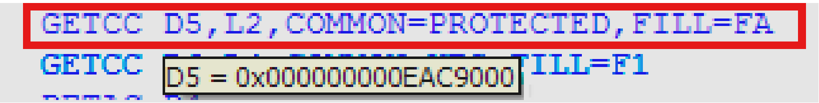
ECB SVA 0x0F148000

File Addr	Subsystem	Module	Status	Holder	Queue Position
D4028191	BSS	QFN8	Holding	Self	0
D4028192	BSS	QFN8	Holding	Self	0
D4028193	BSS	QFN8	Holding	Self	0

See Appendix E for more information

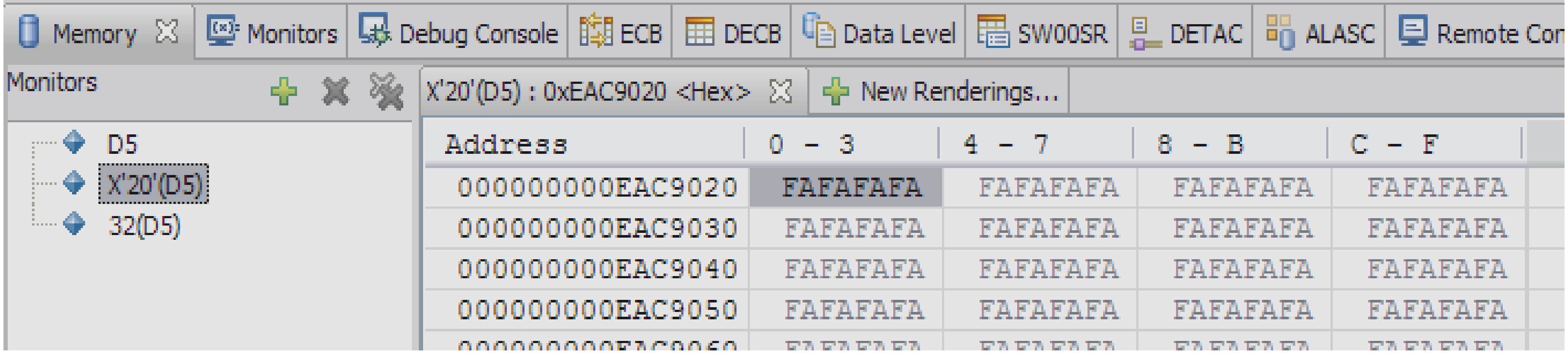
Offset into Dx

- **Provides the ability to specify offsets into data level core blocks to be added as memory monitors in the Memory View**
- **This is similar to our “Offset into Register” expression support where offsets can be provided in hexadecimal or decimal representation**
- **For example:**



```
GETCC D5,L2,COMMON=PROTECTED,FILL=FA
GETCC D5 = 0x000000000EAC9000 ILL=F1
```

(notice D5 now evaluates to an address when hovering over the expression)



```
Address            0 - 3      4 - 7      8 - B      C - F
000000000EAC9020   FAFAFAFA   FAFAFAFA   FAFAFAFA   FAFAFAFA
000000000EAC9030   FAFAFAFA   FAFAFAFA   FAFAFAFA   FAFAFAFA
000000000EAC9040   FAFAFAFA   FAFAFAFA   FAFAFAFA   FAFAFAFA
000000000EAC9050   FAFAFAFA   FAFAFAFA   FAFAFAFA   FAFAFAFA
000000000EAC9060   FAFAFAFA   FAFAFAFA   FAFAFAFA   FAFAFAFA
```

z/TPF Debugger Deliverable Details

Description	z/TPF APAR	TPF Toolkit Level	TPFUG Requirement
Offset into Dx (datalevel) Debug Server Access Control Malloc View Filters Register by SVC Instruction Detail Pane	PJ36679 PUT7	V3.4.6	V09104S V08065F V08036F V08044S V08028F, V08054S, V08052S
XML Generator for C/C++ Record Hold View	PJ37366 PUT7	V3.4.7	V09105S V08032F

z/TPF Debugger Deliverable Details

Description	**z/TPF APAR**	**TPF Toolkit Level**	**TPFUG Requirement**
TPF Code Coverage	PJ37973	Next Release	V08064F

Summary of other New Functionality

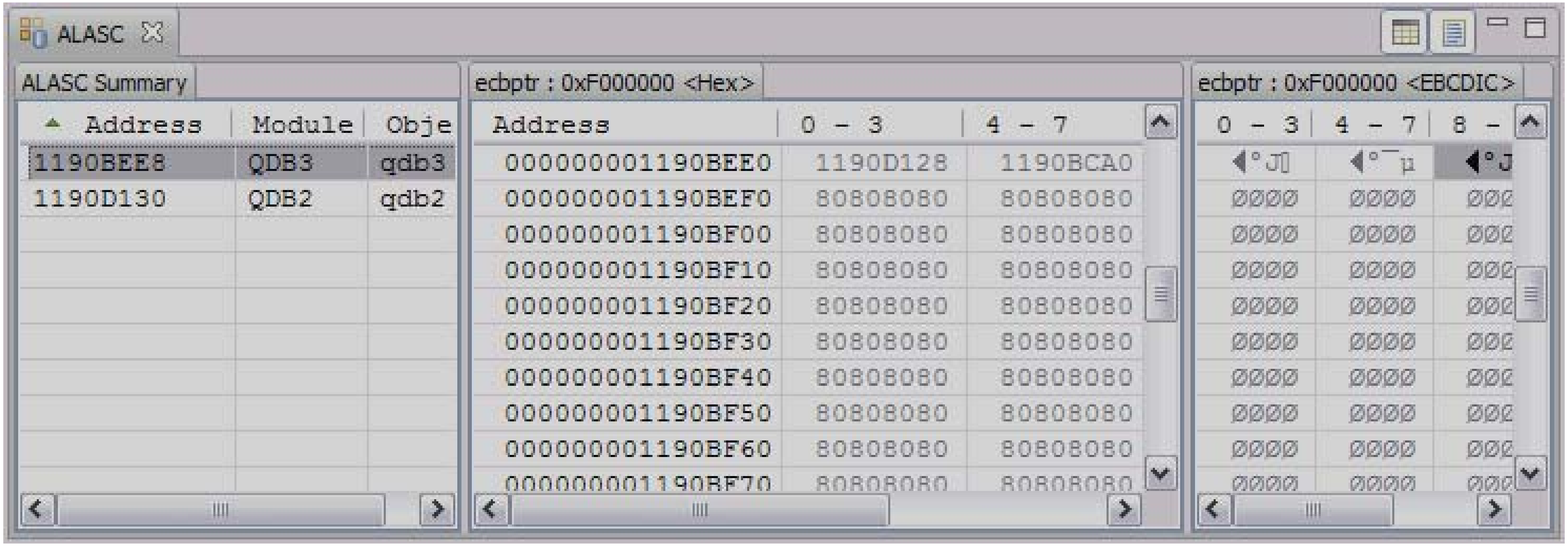
- **ALASC View**
  - Shows all ALASC blocks for the selected ECB.
- **DETAC View**
  - Shows all DETAC blocks for the selected ECB.
- **ECB Launcher**
  - Allows to create a new ECB to run on z/TPF for the program you specify from the TPF Toolkit.
- **Watchpoint Enhancements**
  - The debugger stops when a change is detected at a memory location.
- **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.

Summary of other New Functionality

- **Register by User Defined**
  - Start the debugger on an ECB virtually anywhere in the application based on the conditions registered by the user.
- **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.
- **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.
- **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.

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.



ALASC Summary

Address	Module	Obje
1190BEE8	QDB3	qdb3
1190D130	QDB2	qdb2

ecbptr : 0xF000000 <Hex>

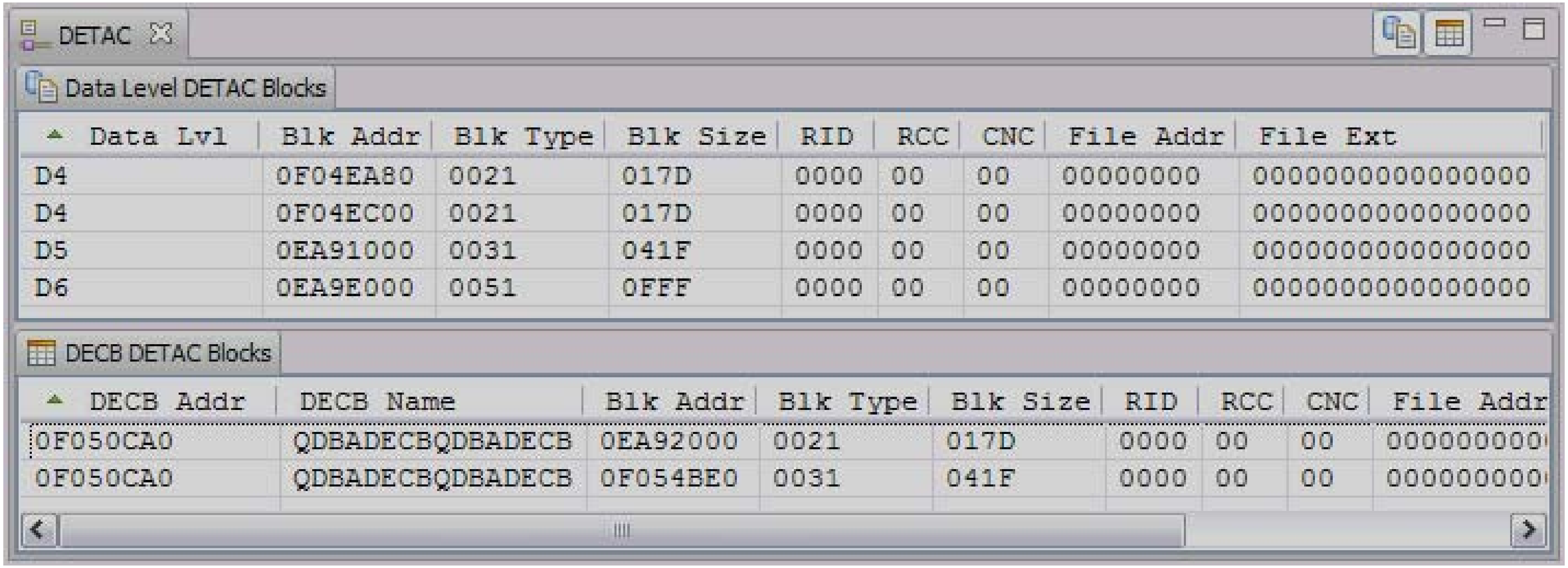
Address	0 - 3	4 - 7
000000001190BEE0	1190D128	1190BCA0
000000001190BEF0	80808080	80808080
000000001190BF00	80808080	80808080
000000001190BF10	80808080	80808080
000000001190BF20	80808080	80808080
000000001190BF30	80808080	80808080
000000001190BF40	80808080	80808080
000000001190BF50	80808080	80808080
000000001190BF60	80808080	80808080
000000001190BF70	80808080	80808080

ecbptr : 0xF000000 <EBCDIC>

0 - 3	4 - 7	8 -
◀°J▯	◀°¯µ	◀°J
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ
ØØØØ	ØØØØ	ØØØ

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.



Data Level DETAC Blocks

Data Lvl	Blk Addr	Blk Type	Blk Size	RID	RCC	CNC	File Addr	File Ext
D4	0F04EA80	0021	017D	0000	00	00	00000000	0000000000000000
D4	0F04EC00	0021	017D	0000	00	00	00000000	0000000000000000
D5	0EA91000	0031	041F	0000	00	00	00000000	0000000000000000
D6	0EA9E000	0051	0FFF	0000	00	00	00000000	0000000000000000

DECB DETAC Blocks

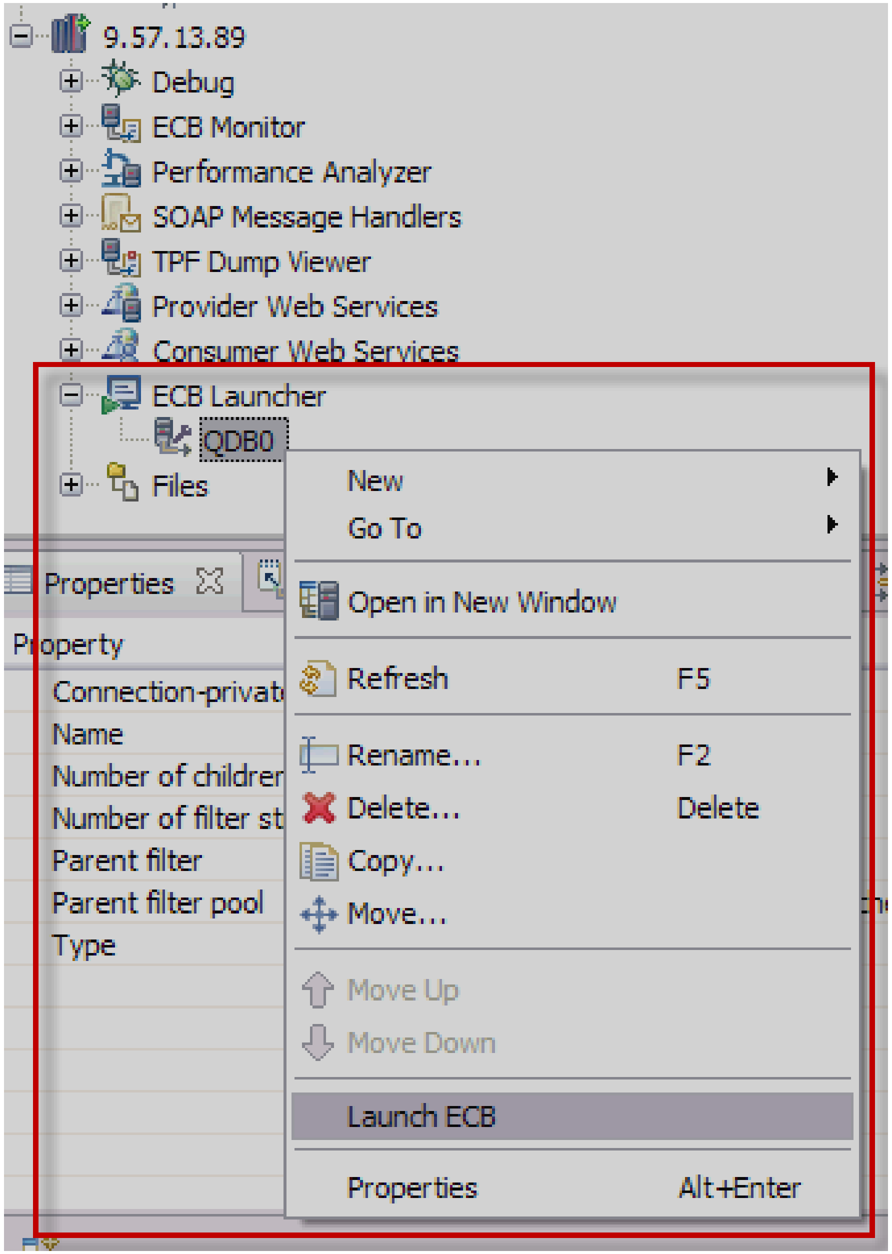
DECB Addr	DECB Name	Blk Addr	Blk Type	Blk Size	RID	RCC	CNC	File Addr
0F050CA0	QDBADECBQDBADECB	0EA92000	0021	017D	0000	00	00	000000000
0F050CA0	QDBADECBQDBADECB	0F054BE0	0031	041F	0000	00	00	000000000

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.

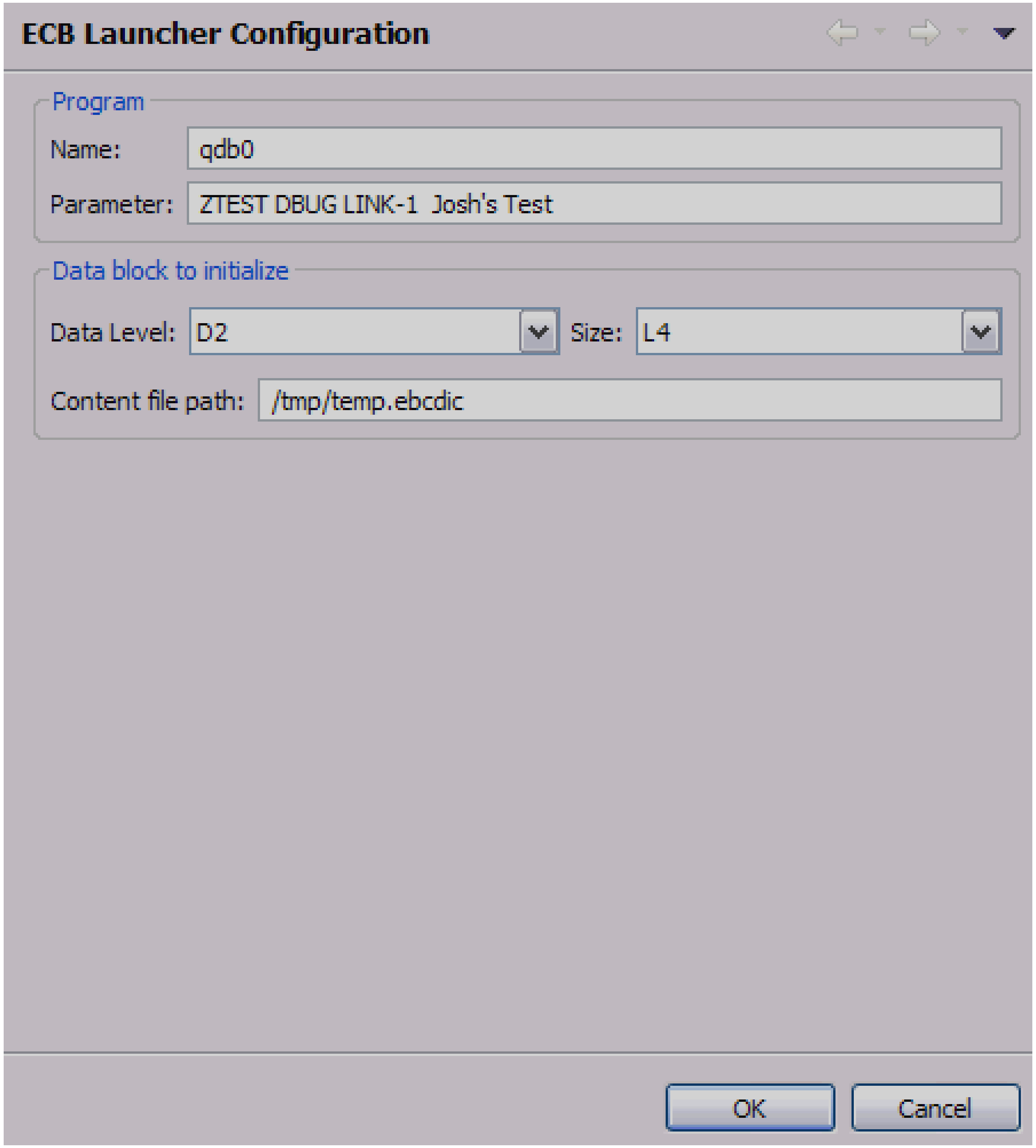
ECB Launcher

- 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.



ECB Launcher Configuration

Program

Name: qdb0

Parameter: ZTEST DBUG LINK-1 Josh's Test

Data block to initialize

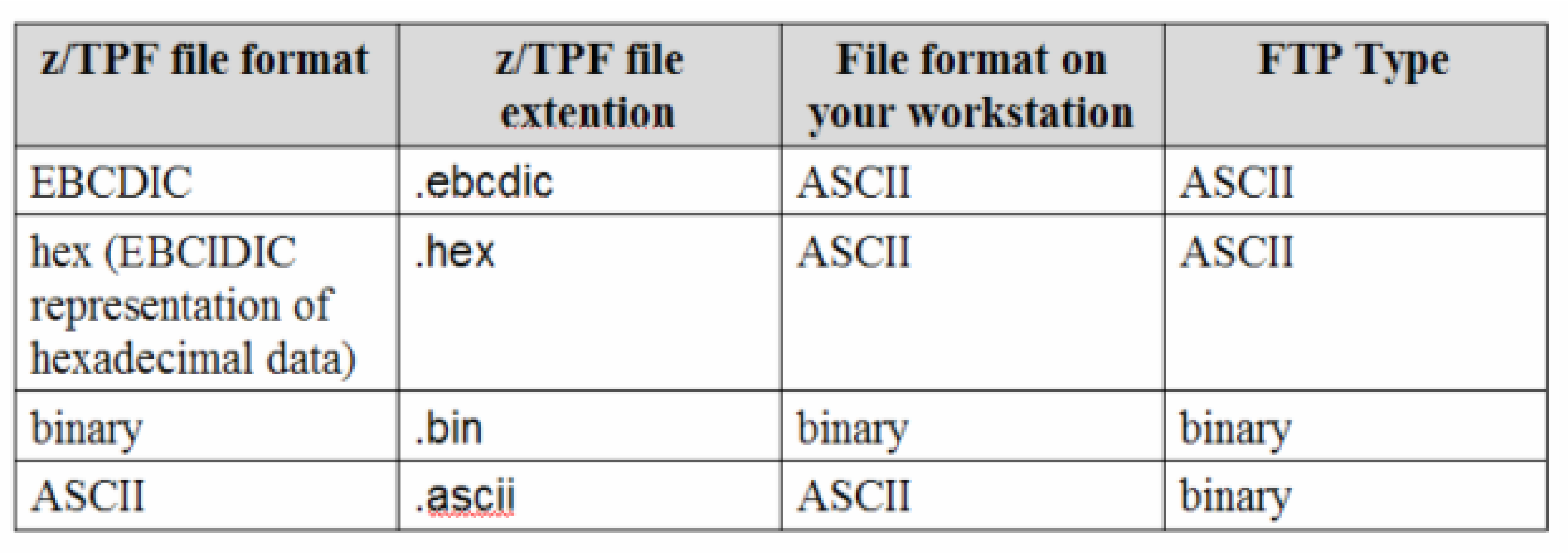
Data Level: D2

Size: L4

Content file path: /tmp/temp.ebcdic

ECB Launcher

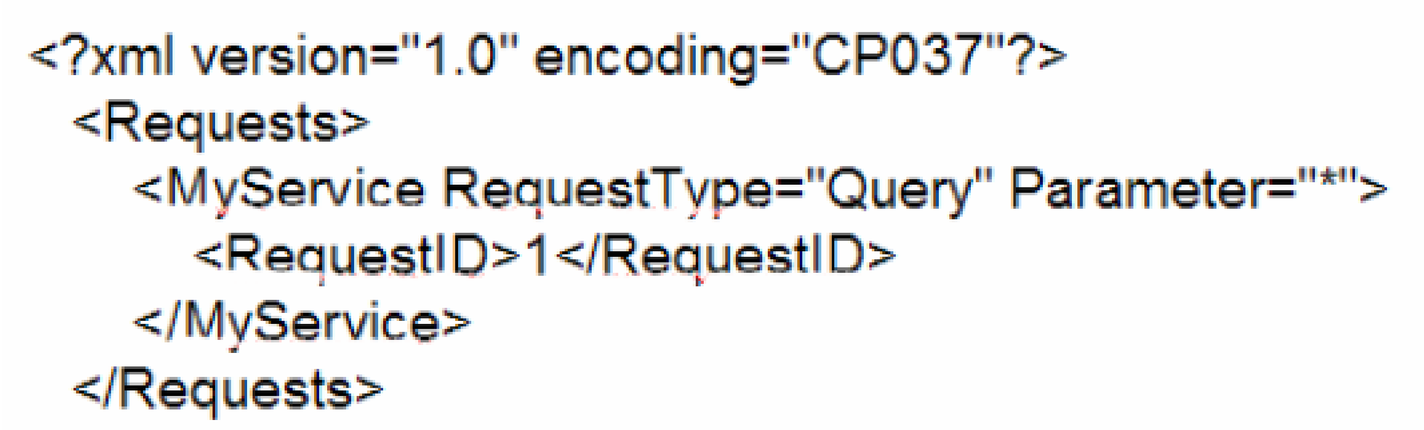
- Data Level content file formats supported (Table indicates how the files must be moved to the TPF file system):



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

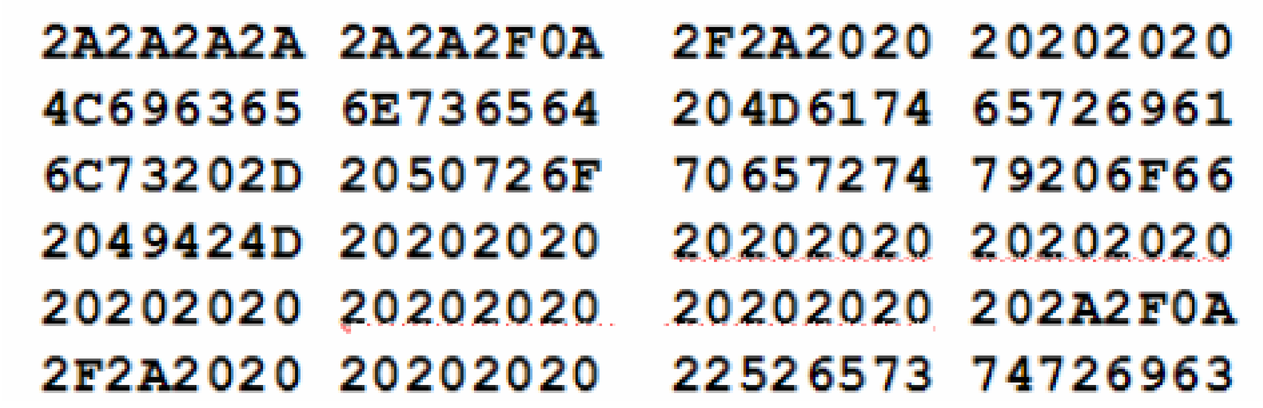
ECB Launcher

- 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>
```

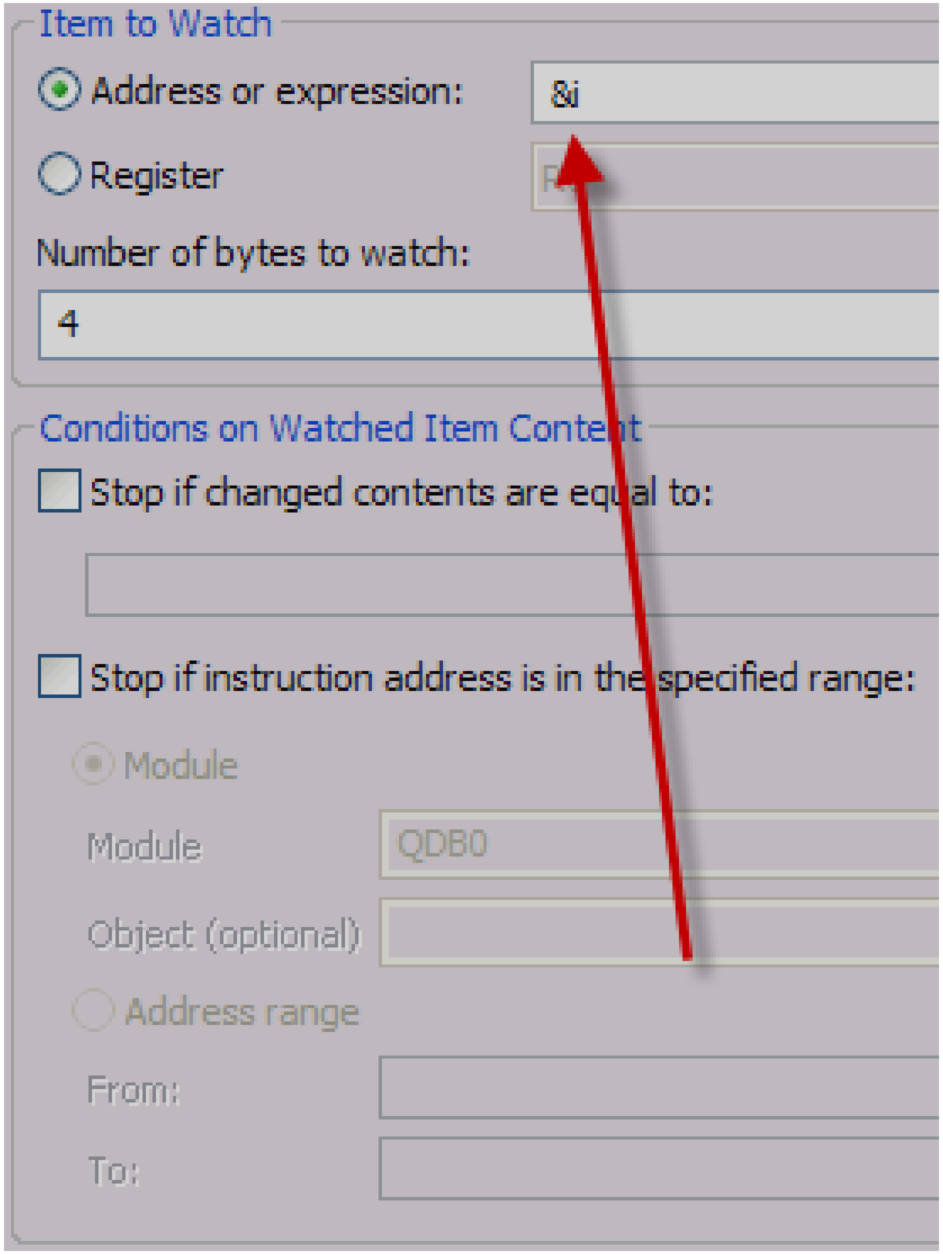
  - hex example (data copied from a dump into a text file):



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

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 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 <u>any</u> value. For example the statement: i = 5;



Item to Watch

● Address or expression: &i

○ Register R

Number of bytes to watch:

4

Conditions on Watched Item Content

□ Stop if changed contents are equal to:

□ Stop if instruction address is in the specified range:

● Module

Module QDB0

Object (optional)

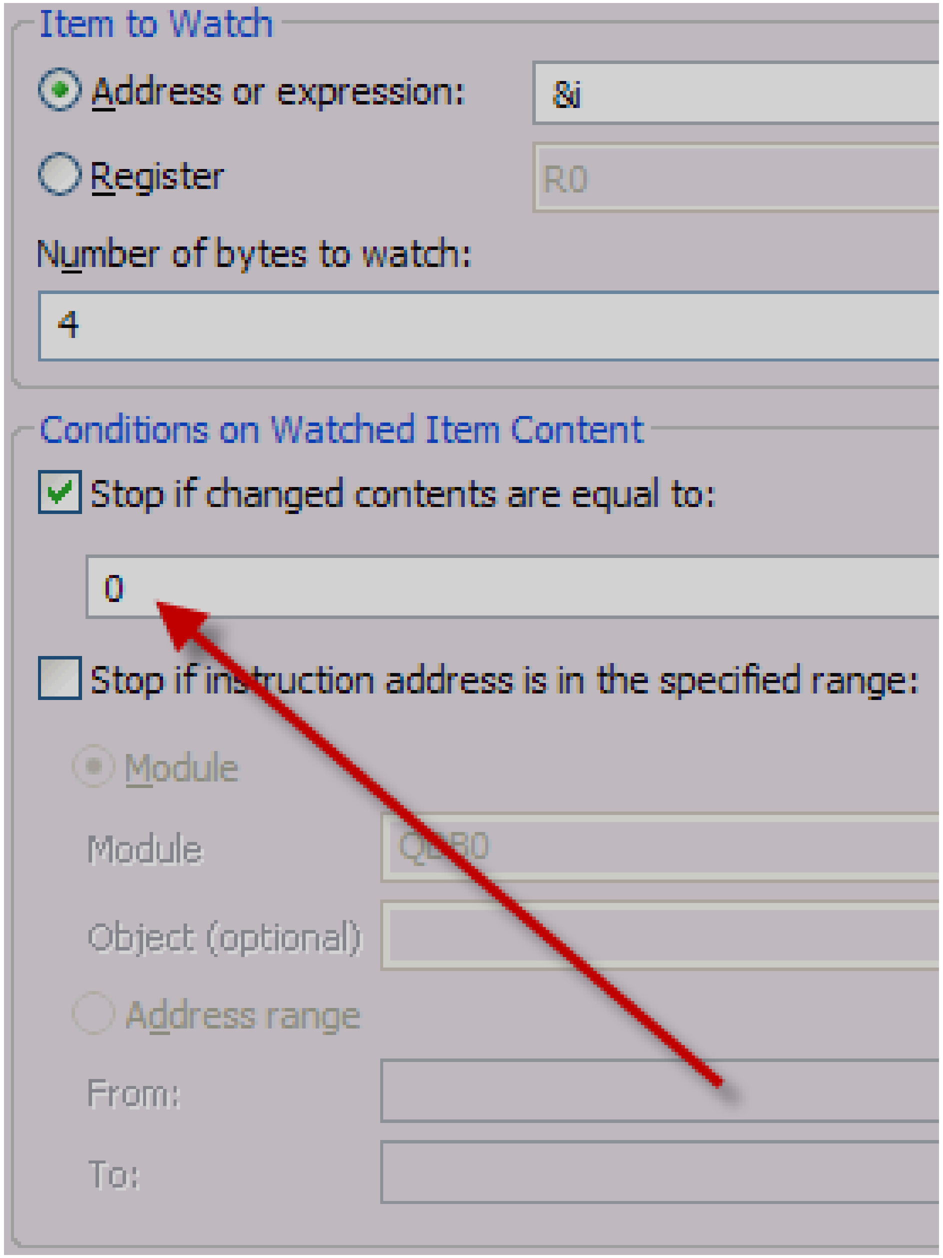
○ Address range

From:

To:

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: &i

○ 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)

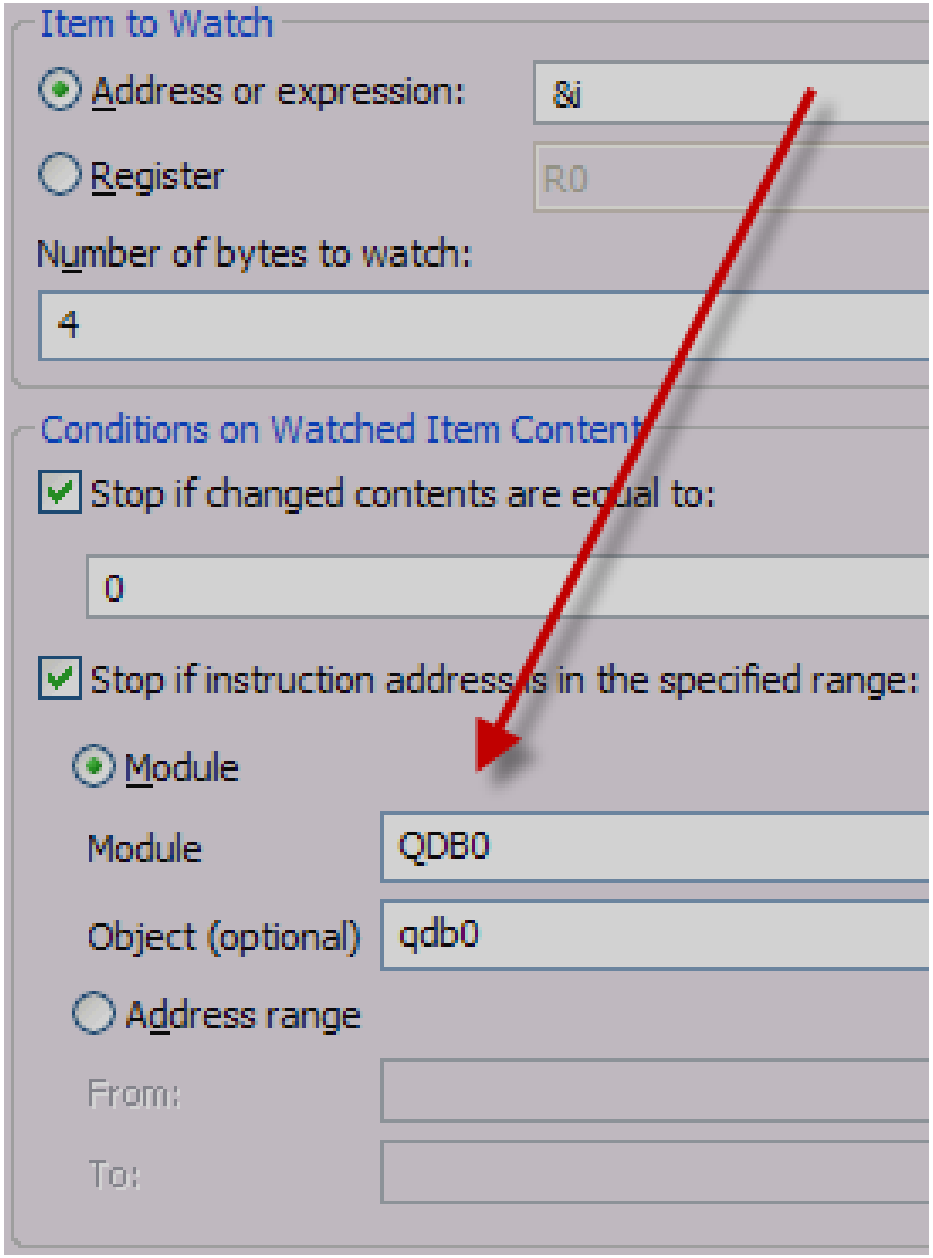
○ Address range

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: &i

○ 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

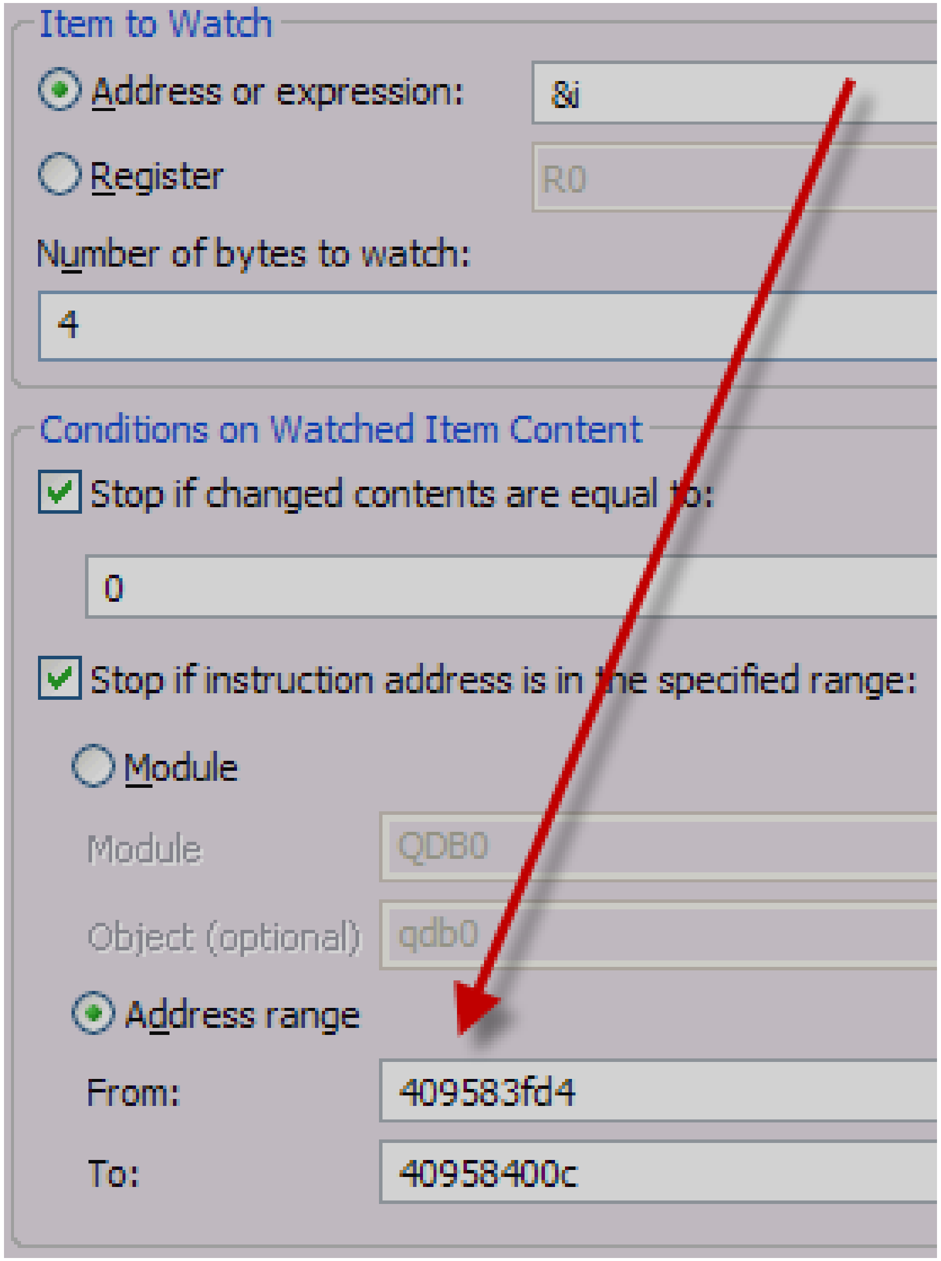
○ Address range

From:

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: &i

○ 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

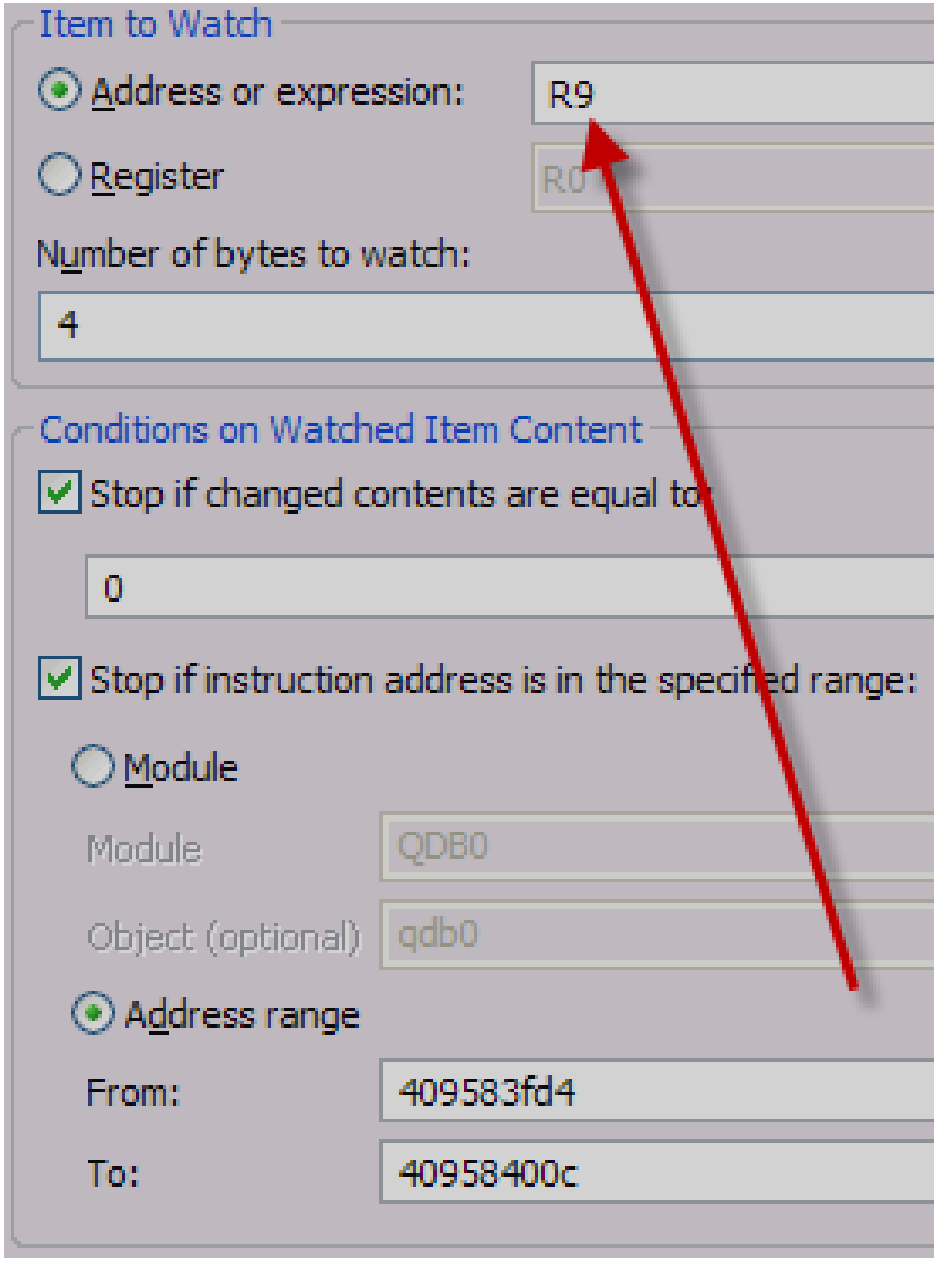
● Address range

From: 409583fd4

To: 40958400c

Watchpoint Enhancements

- 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

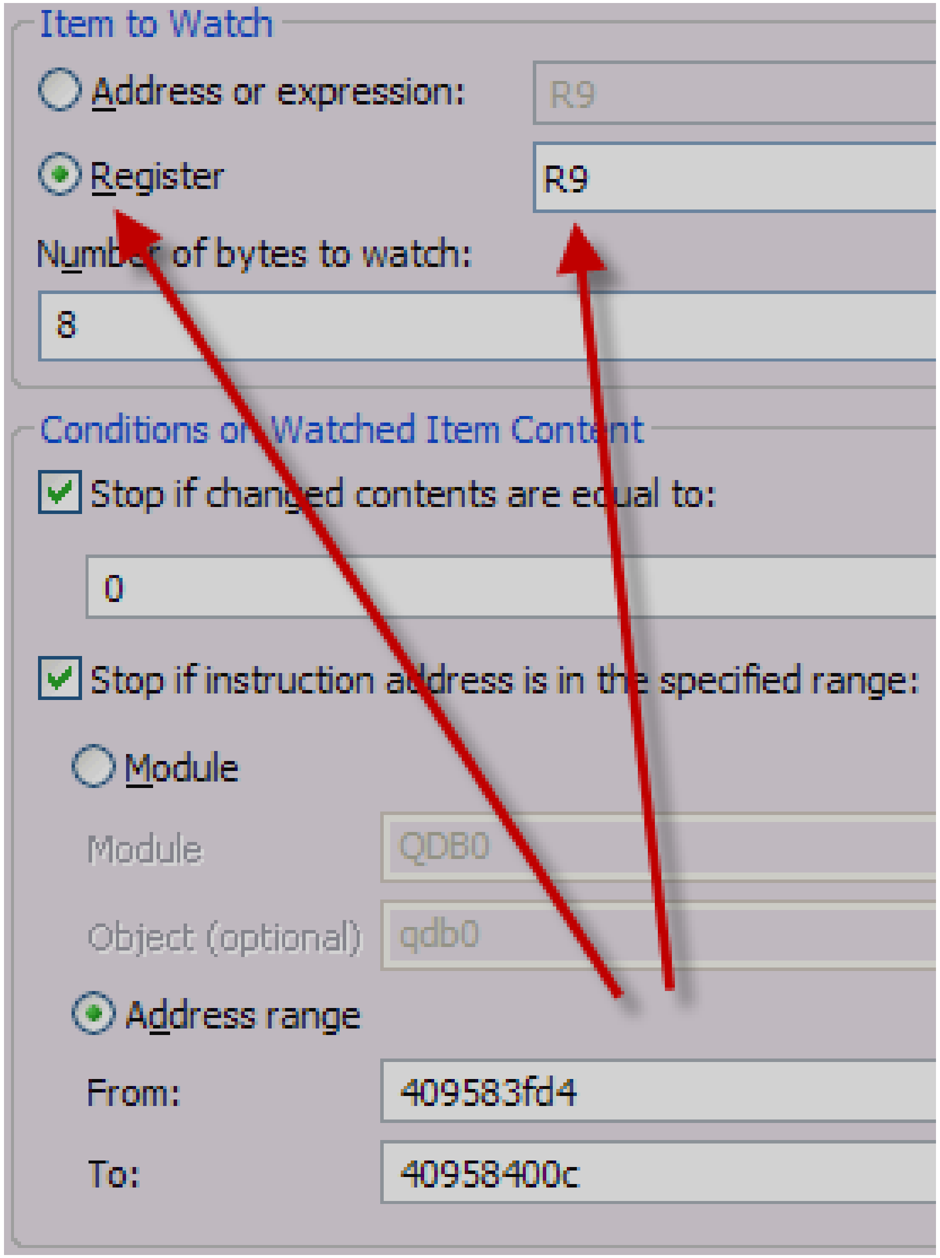
● Address range

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

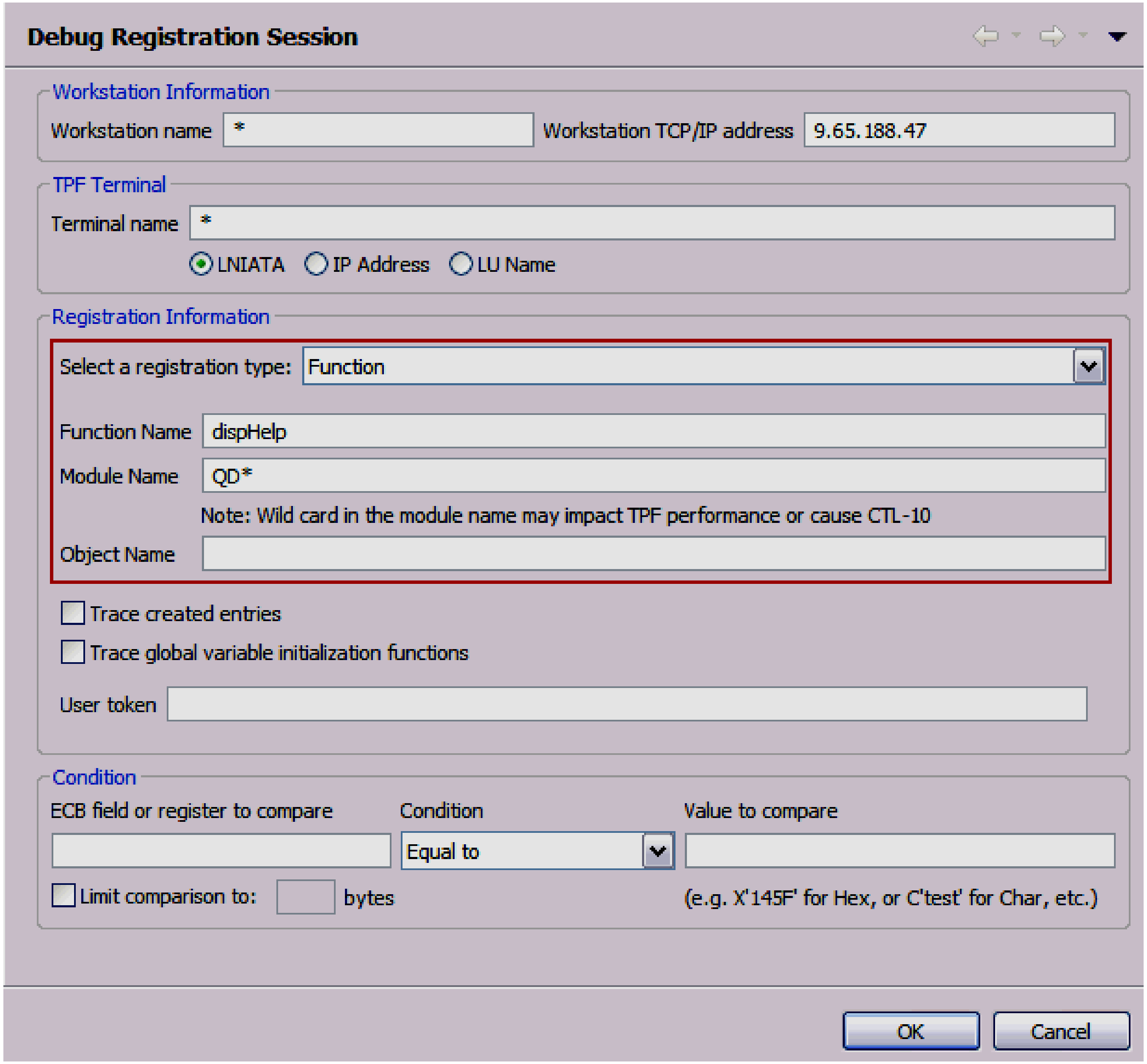
● Address range

From: 409583fd4

To: 40958400c

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: Equal to

Value to compare:

□ Limit comparison to: bytes

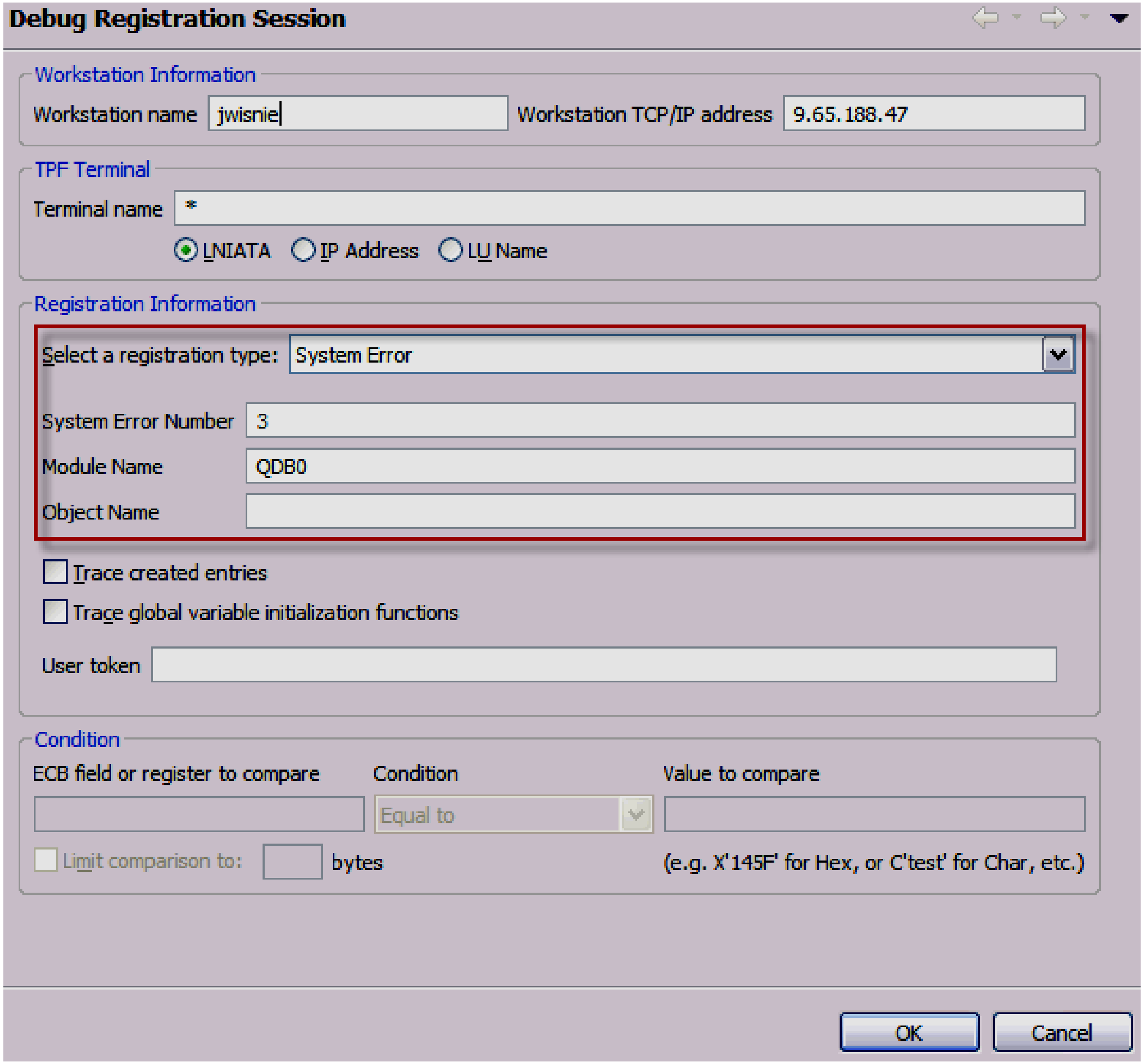
(e.g. X'145F' for Hex, or C'test' for Char, etc.)

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: _ZN22IVAExceptionBreakpointC1E9IVAString**
- **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: jwisnie

Workstation TCP/IP address: 9.65.188.47

TPF Terminal

Terminal name: *

● LNIATA ○ IP Address ○ LU Name

Registration Information

Select a registration type: System Error

System Error Number: 3

Module Name: QDB0

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	

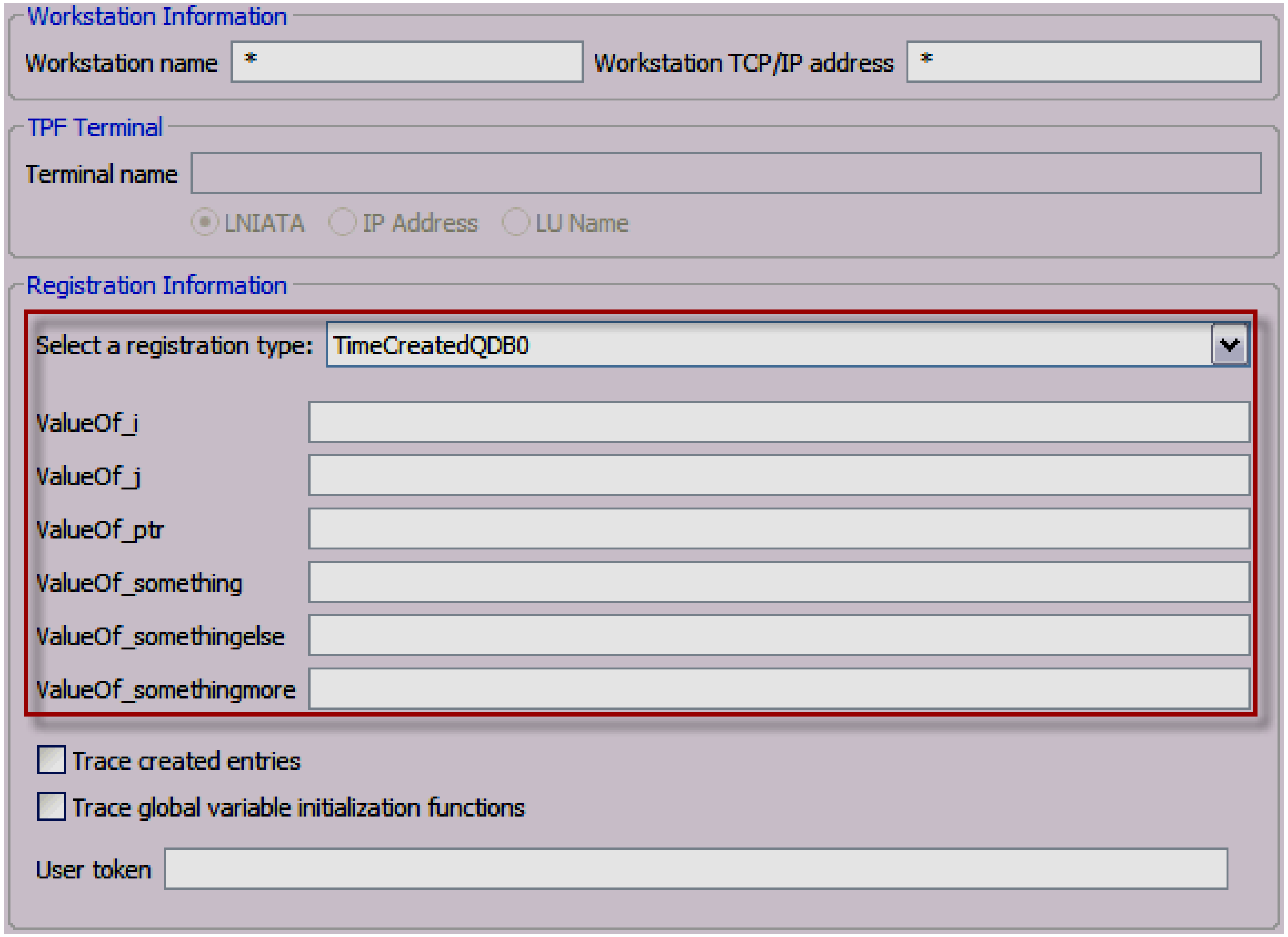
□ Limit comparison to: bytes (e.g. X'145F' for Hex, or C'test' for Char, etc.)

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: TimeCreatedQDB0

ValueOf_i:

ValueOf_j:

ValueOf_ptr:

ValueOf_something:

ValueOf_somethingelse:

ValueOf_somethingmore:

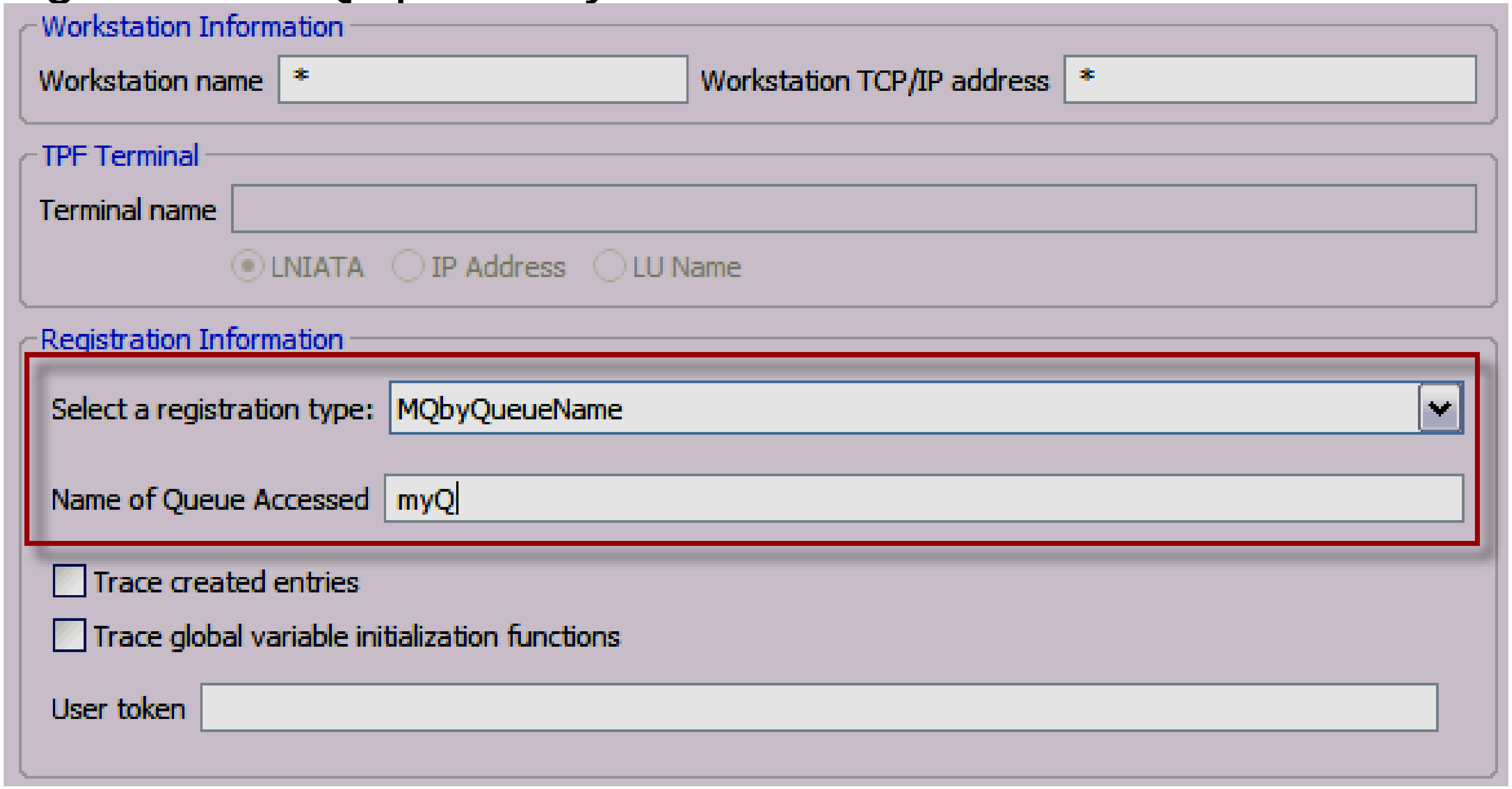
□ Trace created entries

□ 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: MQbyQueueName

Name of Queue Accessed: myQ

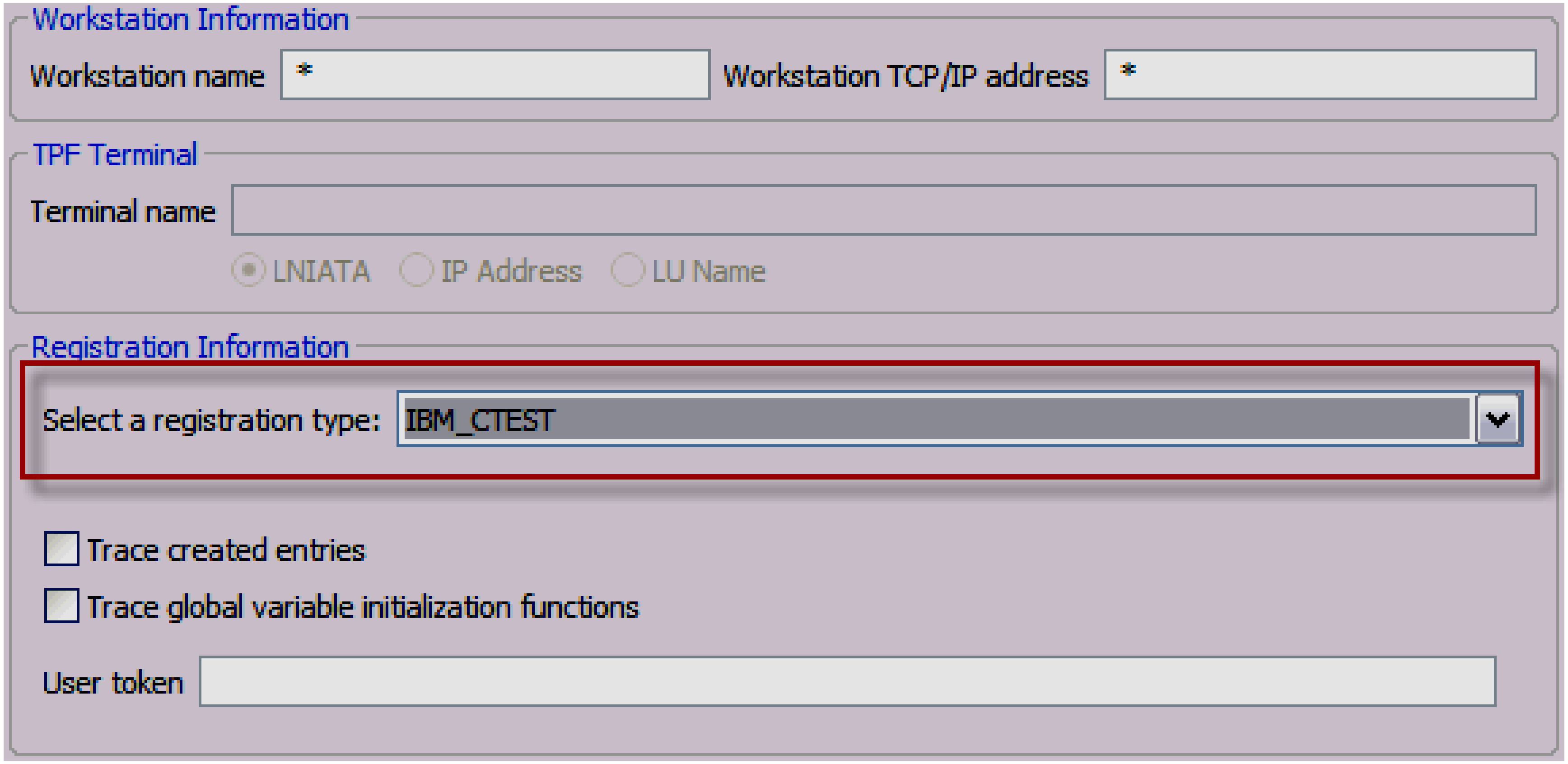
□ Trace created entries

□ Trace global variable initialization functions

User token:

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: IBM_CTEST

□ Trace created entries

□ Trace global variable initialization functions

User token:

Register by User Defined

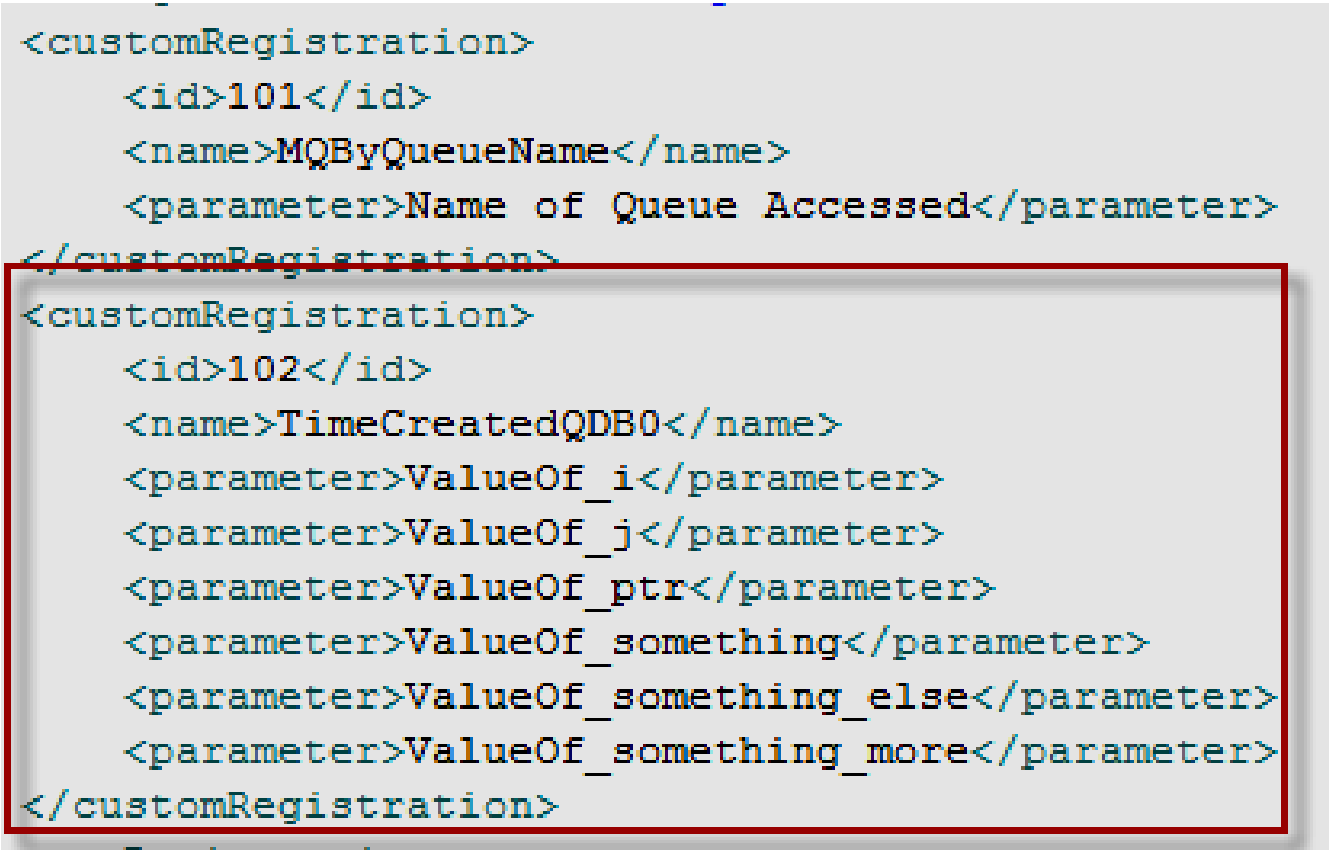
- **How is register by user defined setup by an administrator?**

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.**

Register by User Defined

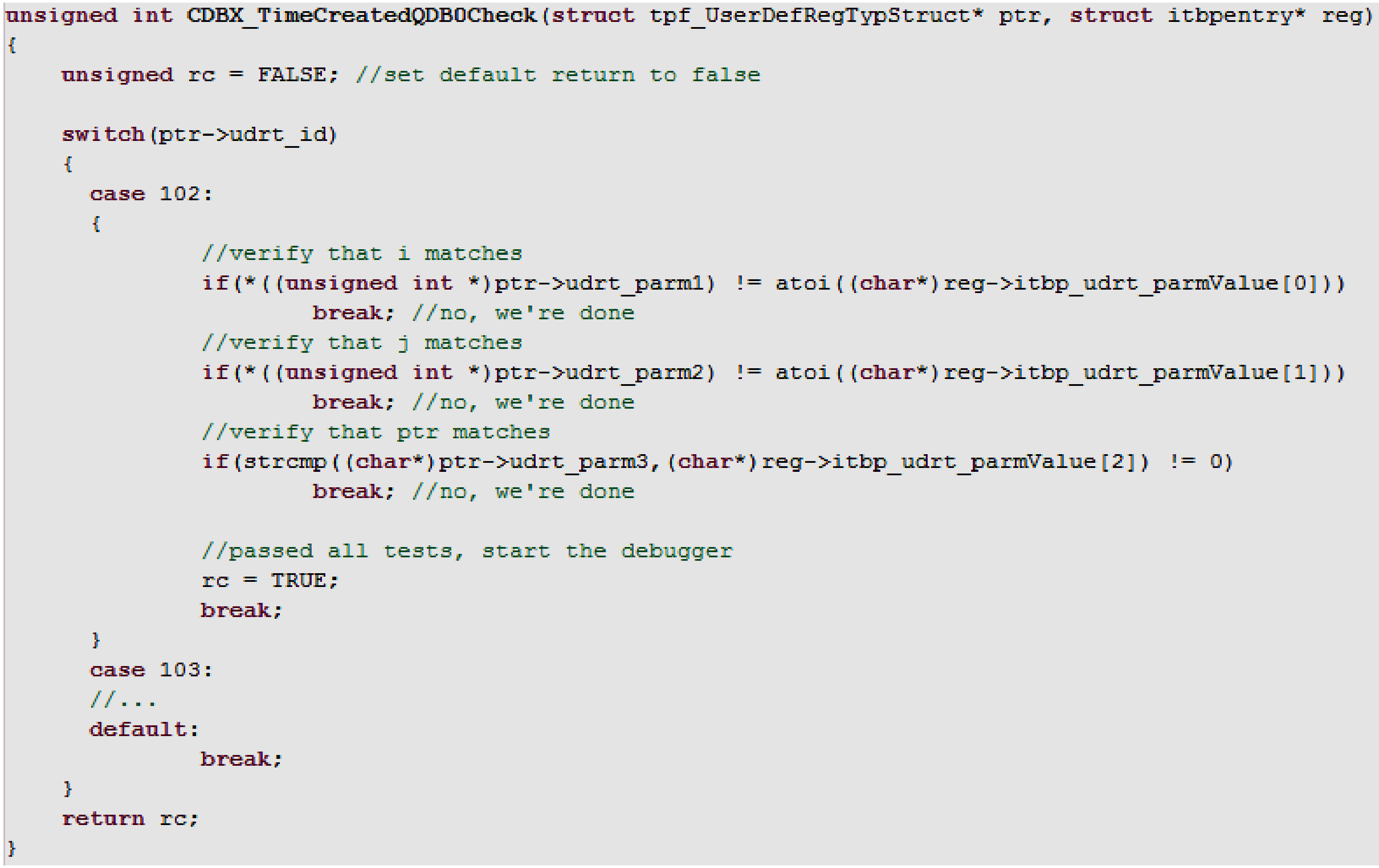
1. **Modify the file <TPF Toolkit install dir>\Config\TPFSHARE\Debug Registration/customDebugRegTypes.xml**
   - Ids 101-255 are for customer use (0-100 are reserved for IBM)
   - Specify the registration name and up to 6 parameter names
2. **Restart the TPF Toolkit**



```
<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>
```

Register by User Defined

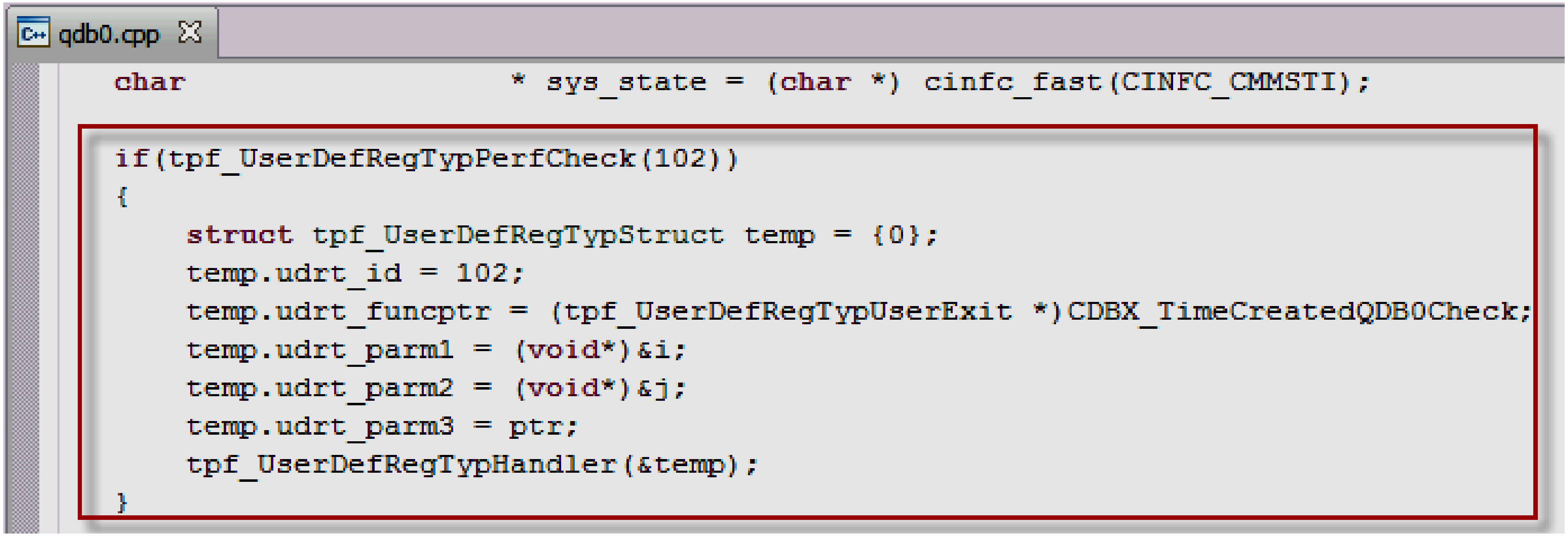
3. **Implement the resolving function to test application state against the user registered values**



```
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;
}
```

Register by User Defined

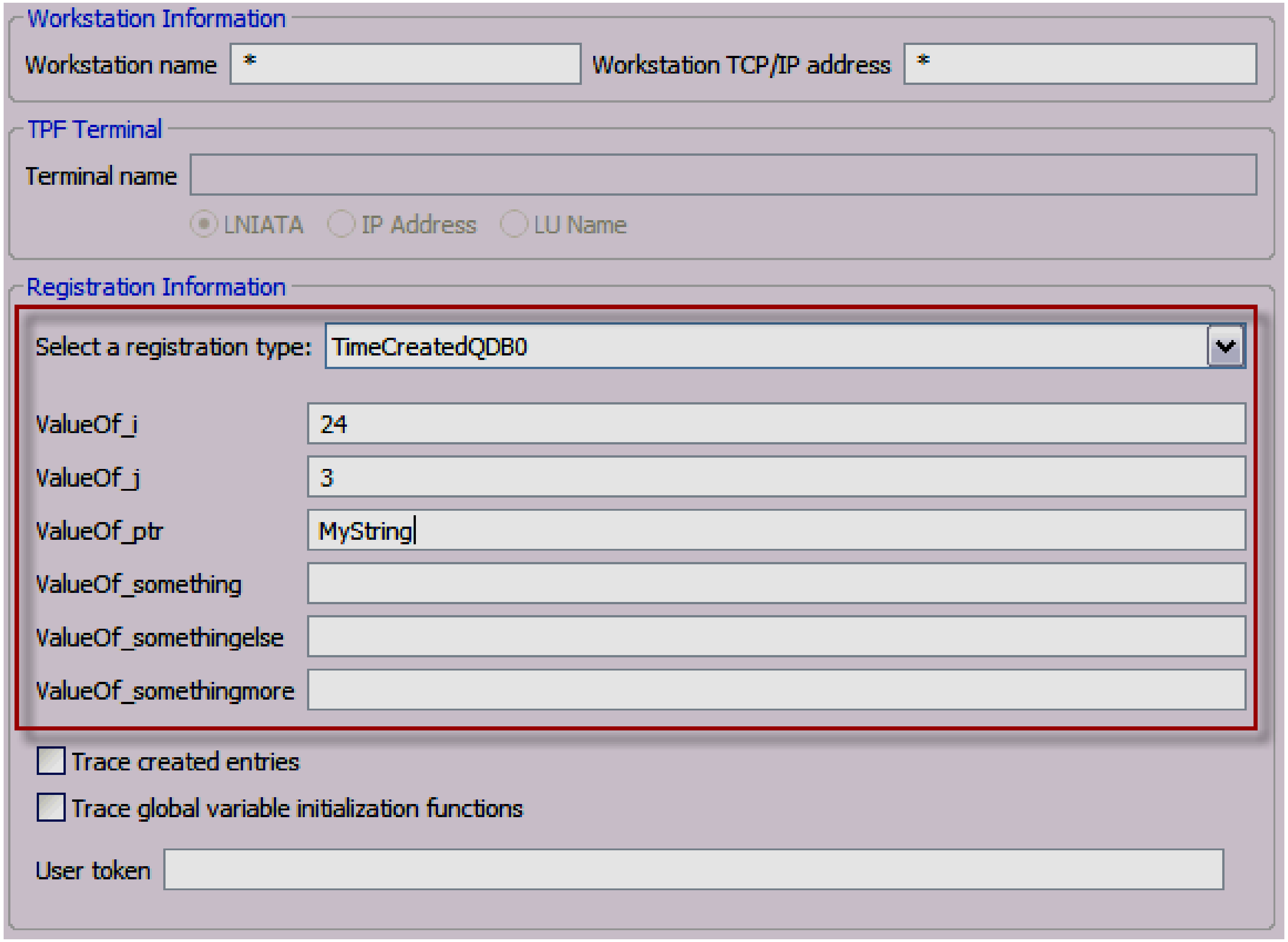
4. **Update the application code to call User Defined Registration handler, passing in the resolving function to use.**



```
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);
}
```

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: TimeCreatedQDB0

ValueOf_i: 24

ValueOf_j: 3

ValueOf_ptr: MyString

ValueOf_something:

ValueOf_somethingelse:

ValueOf_somethingmore:

□ Trace created entries

□ Trace global variable initialization functions

User token:

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.**

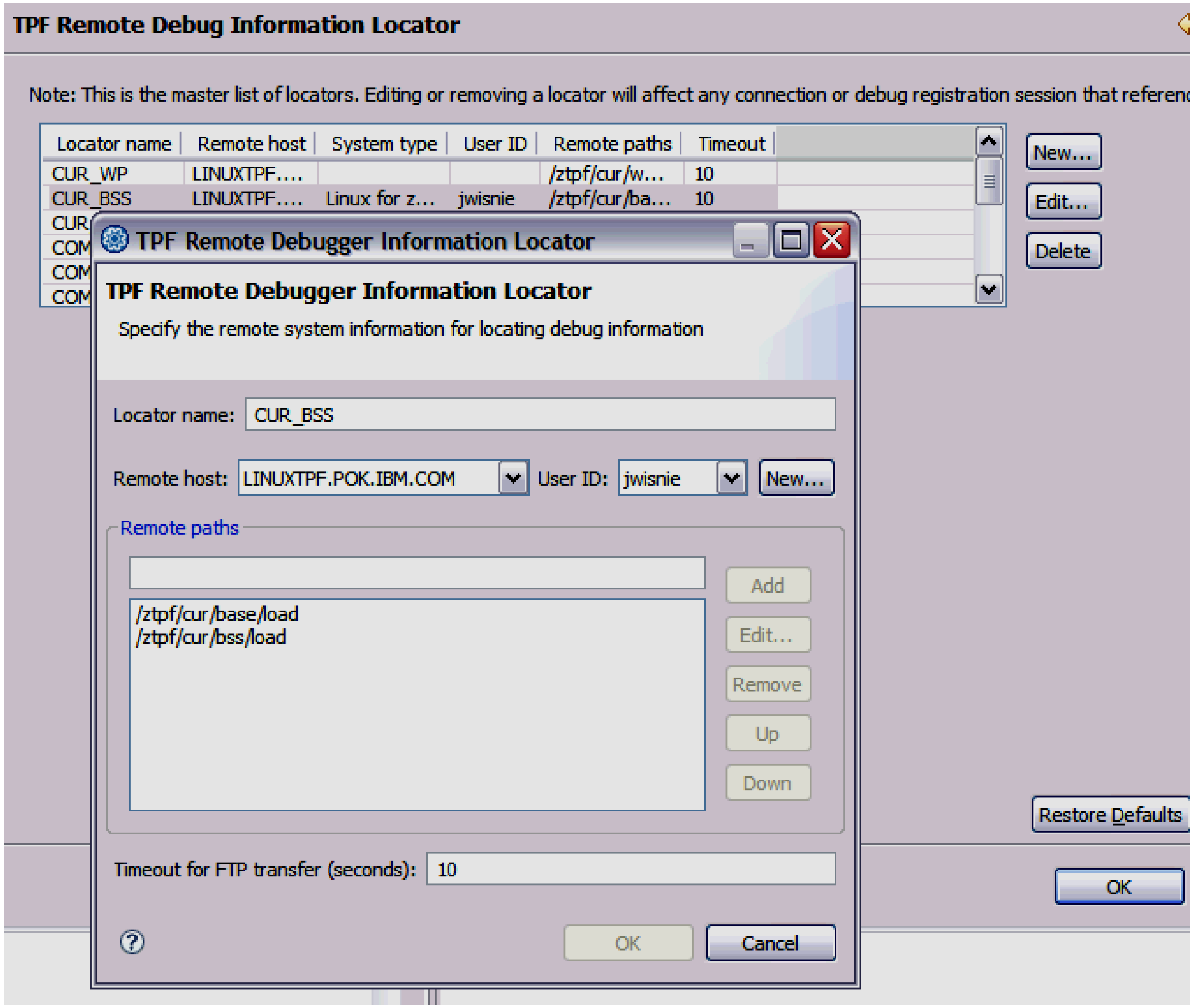
Remote Debug Info

- **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.**



TPF Remote Debug Information Locator

Note: This is the master list of locators. Editing or removing a locator will affect any connection or debug registration session that referen

Locator name	Remote host	System type	User ID	Remote paths	Timeout
CUR_WP	LINUXTPF....			/ztpf/cur/w...	10
CUR_BSS	LINUXTPF....	Linux for z...	jwisnie	/ztpf/cur/ba...	10
CUR					
COM					
COM					
COM					

TPF Remote Debugger Information Locator

TPF Remote Debugger Information Locator

Specify the remote system information for locating debug information

Locator name: CUR_BSS

Remote host: LINUXTPF.POK.IBM.COM User ID: jwisnie

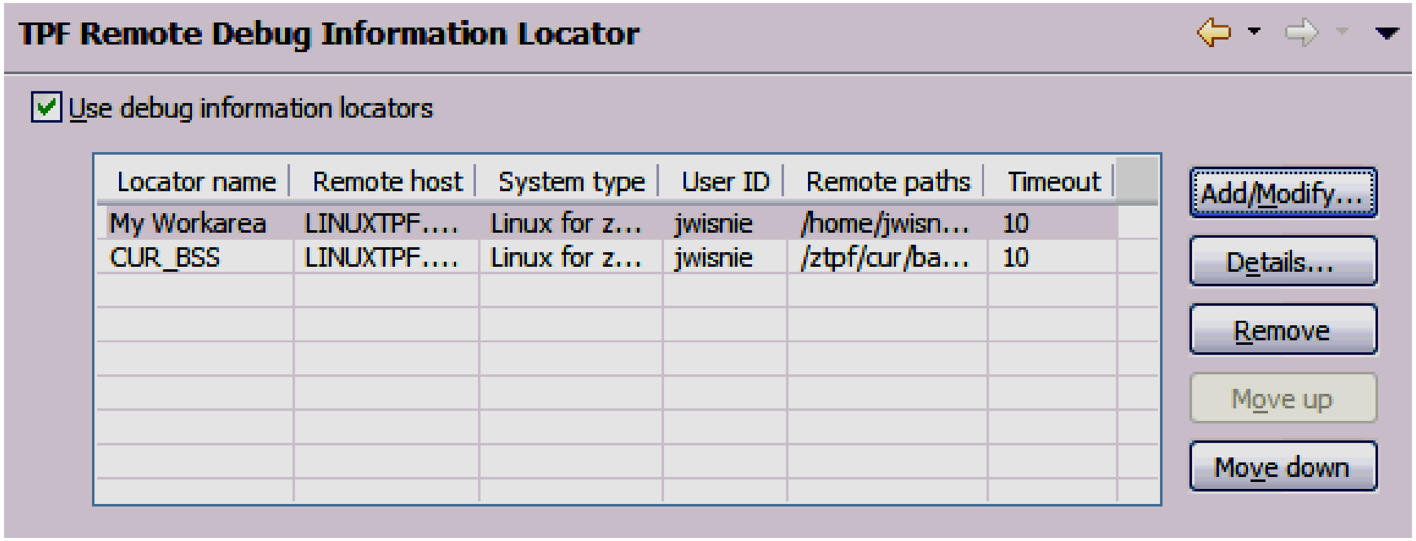
Remote paths

/ztpf/cur/base/load
/ztpf/cur/bss/load

Timeout for FTP transfer (seconds): 10

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.**



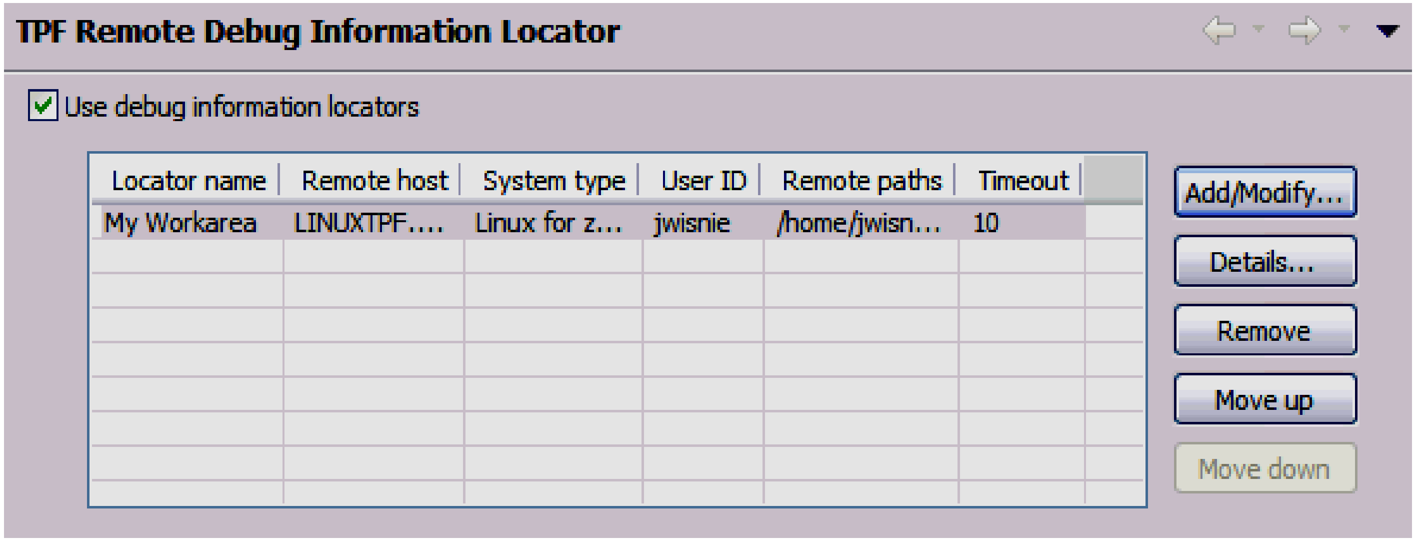
TPF Remote Debug Information Locator

☑ Use debug information locators

Locator name	Remote host	System type	User ID	Remote paths	Timeout
My Workarea	LINUXTPF....	Linux for z...	jwisnie	/home/jwisn...	10
CUR_BSS	LINUXTPF....	Linux for z...	jwisnie	/ztpf/cur/ba...	10

Remote Debug Info (Debug Session)

3. **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.**



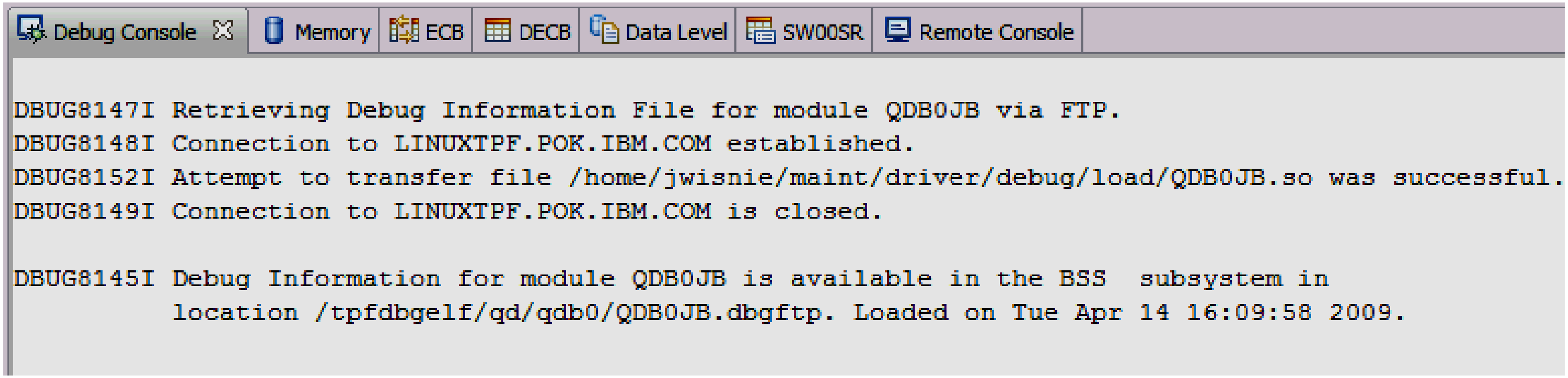
TPF Remote Debug Information Locator

☑ Use debug information locators

Locator name	Remote host	System type	User ID	Remote paths	Timeout
My Workarea	LINUXTPF....	Linux for z...	jwisnie	/home/jwisn...	10

Remote Debug Info (Debug Session)

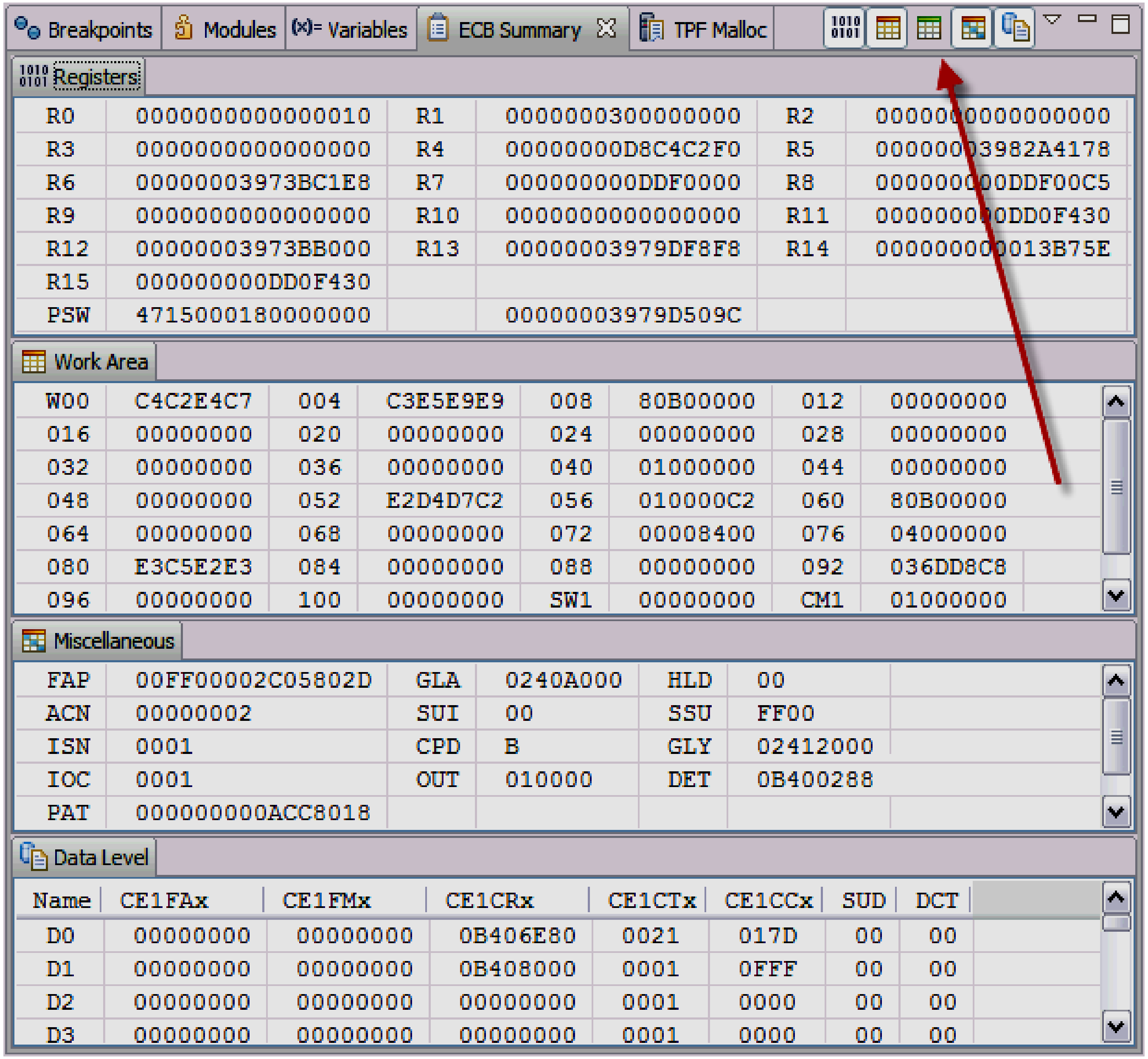
- **Debug console messages are now sent to the TPF Toolkit to notify the user if debug information could be located and what debug information file was used.**



```
DBUG8147I Retrieving Debug Information File for module QDB0JB via FTP.
DBUG8148I Connection to LINUXTPF.POK.IBM.COM established.
DBUG8152I Attempt to transfer file /home/jwisnie/maint/driver/debug/load/QDB0JB.so was successful.
DBUG8149I Connection to LINUXTPF.POK.IBM.COM is closed.

DBUG8145I 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.
```

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



Registers

R0	0000000000000010	R1	0000000300000000	R2	0000000000000000
R3	0000000000000000	R4	00000000D8C4C2F0	R5	00000003982A4178
R6	00000003973BC1E8	R7	000000000DDF0000	R8	000000000DDF00C5
R9	0000000000000000	R10	0000000000000000	R11	000000000DD0F430
R12	00000003973BB000	R13	00000003979DF8F8	R14	000000000013B75E
R15	000000000DD0F430				
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

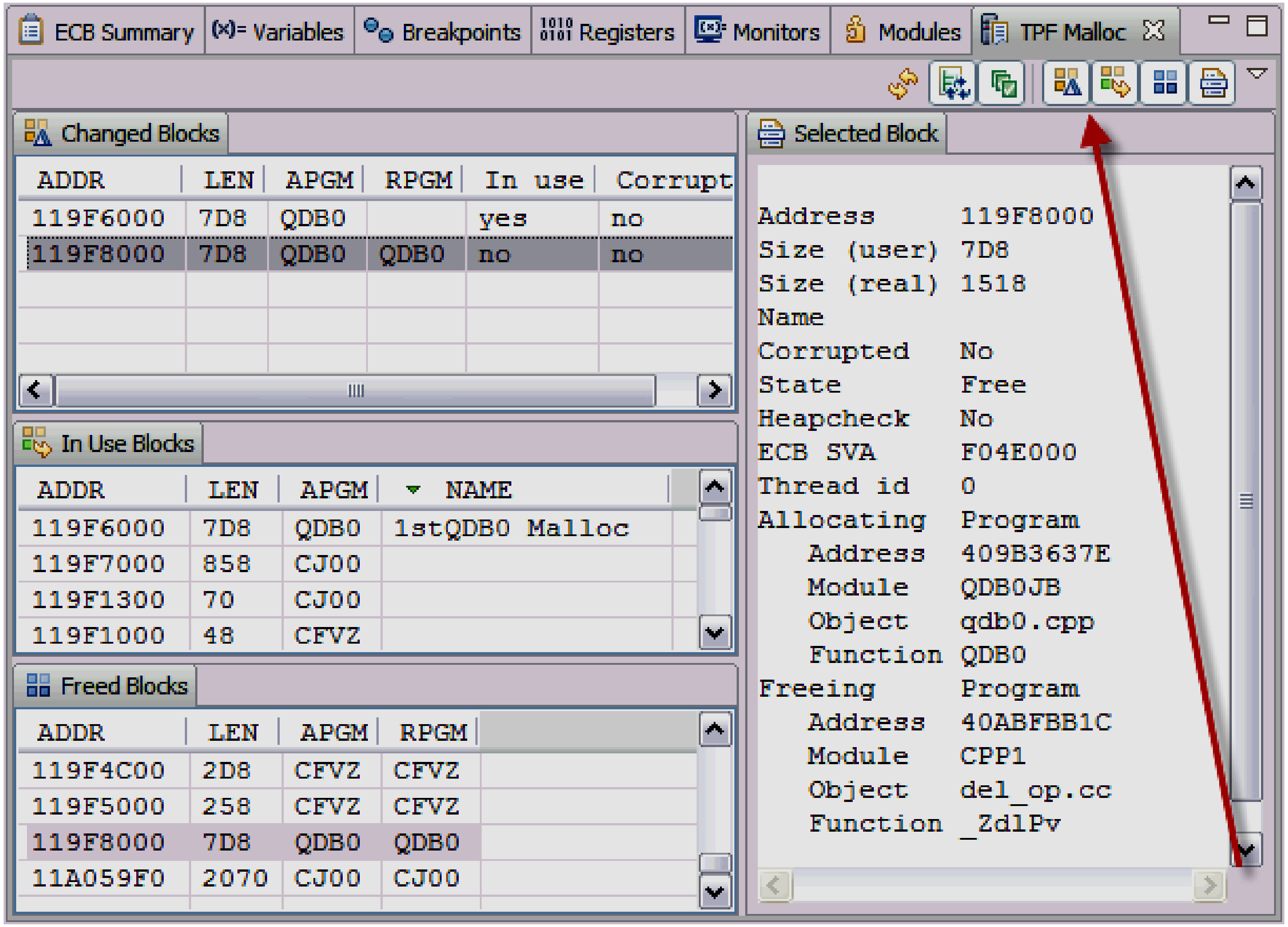
FAP	00FF00002C05802D	GLA	0240A000	HLD	00
ACN	00000002	SUI	00	SSU	FF00
ISN	0001	CPD	B	GLY	02412000
IOC	0001	OUT	010000	DET	0B400288
PAT	000000000ACC8018				

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.**



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

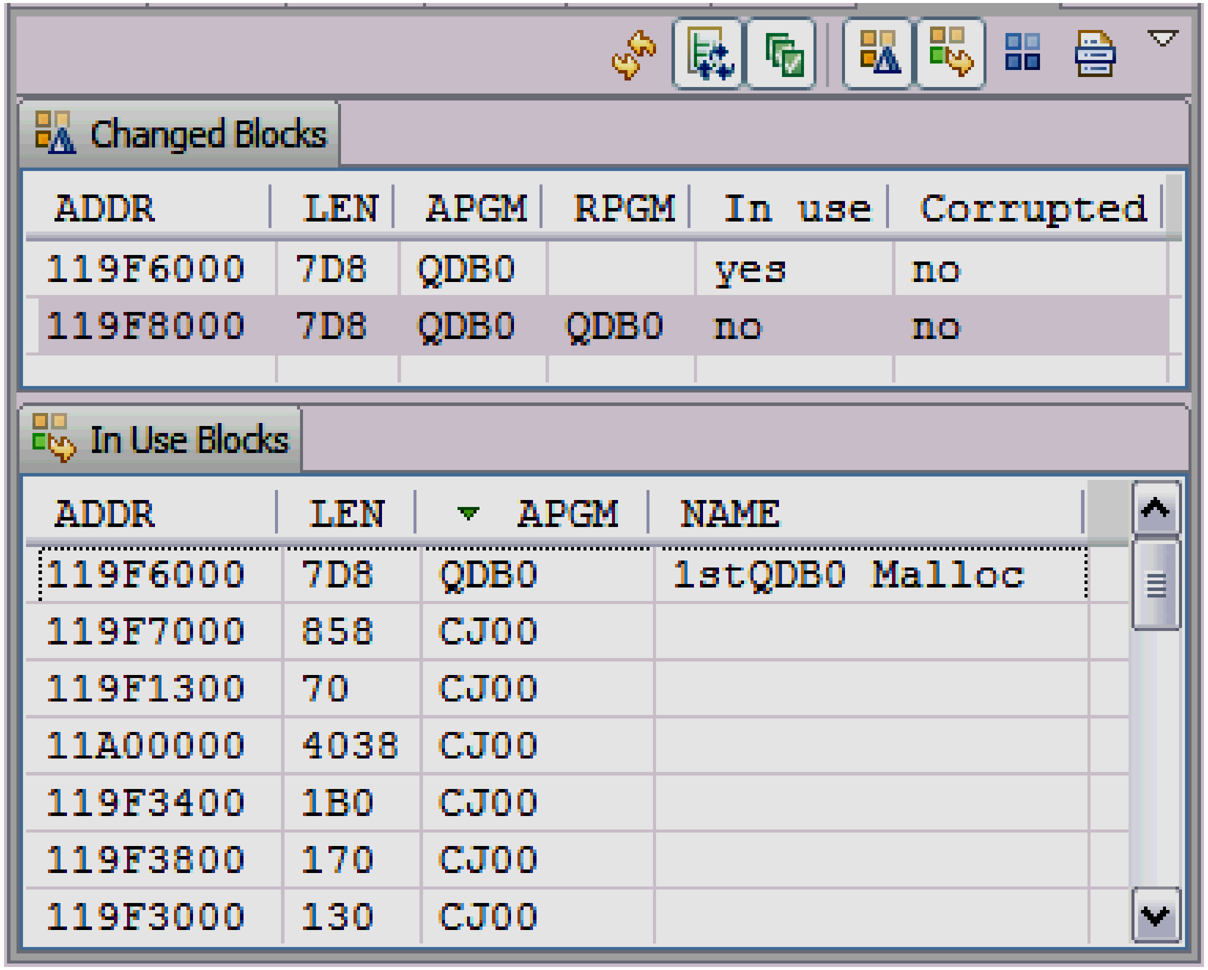
ADDR	LEN	APGM	RPGM
119F4C00	2D8	CFVZ	CFVZ
119F5000	258	CFVZ	CFVZ
119F8000	7D8	QDB0	QDB0
11A059F0	2070	CJ00	CJ00

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

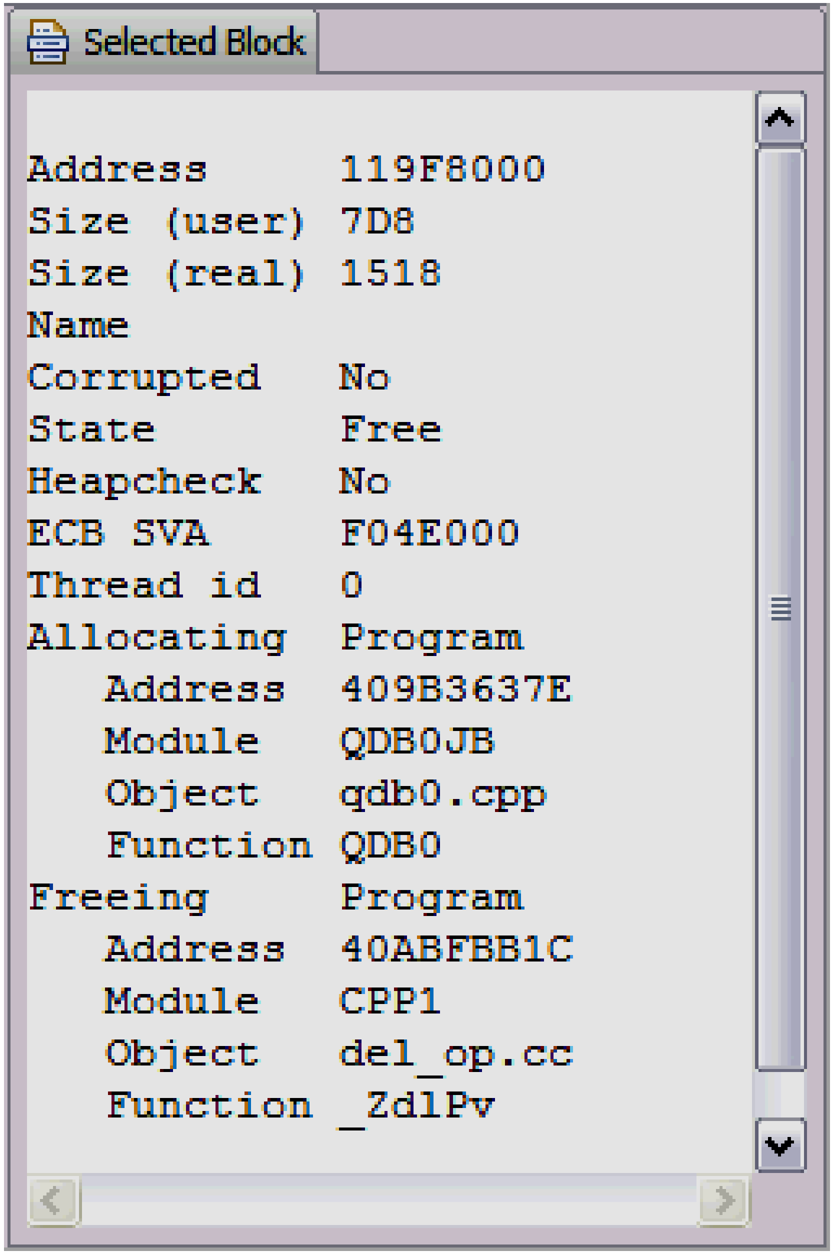
ADDR	LEN	APGM	RPGM	In use	Corrupted
119F6000	7D8	QDB0		yes	no
119F8000	7D8	QDB0	QDB0	no	no

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.**



```
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.**

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.**

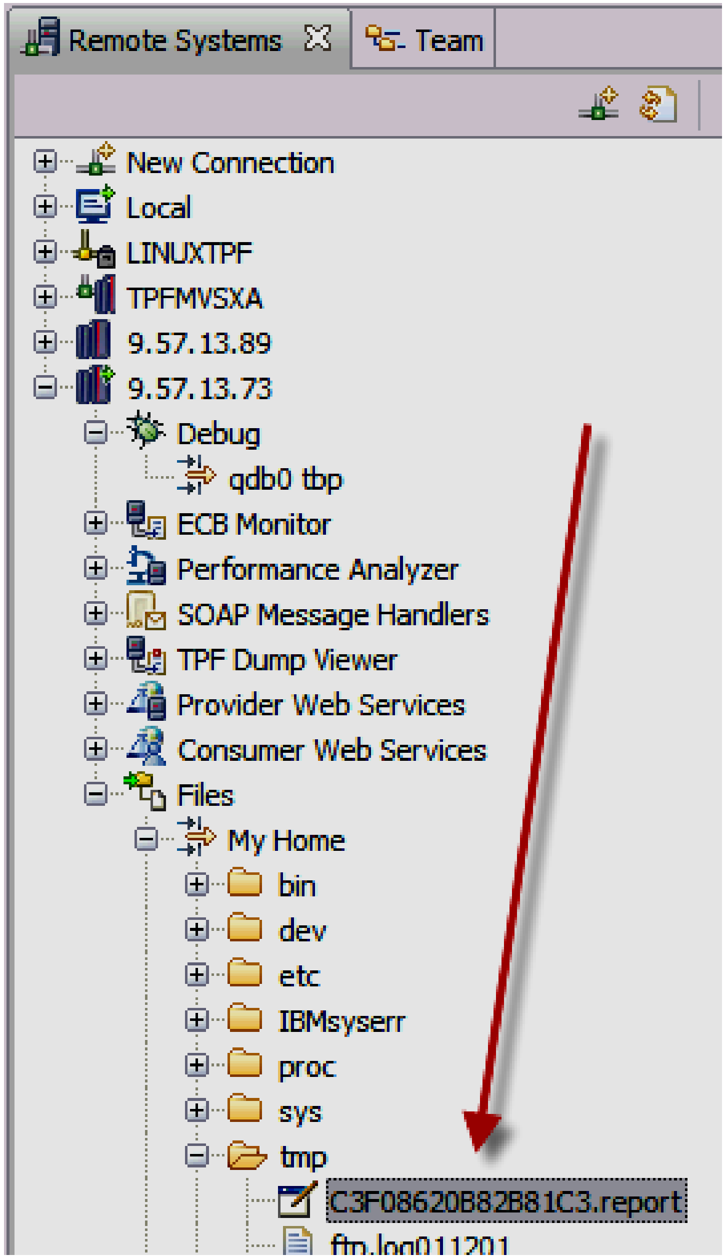
- **A new TRLOG parameter has been provided to produce the trace log file in text format with the extension .report such that post processing is not required.**

  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

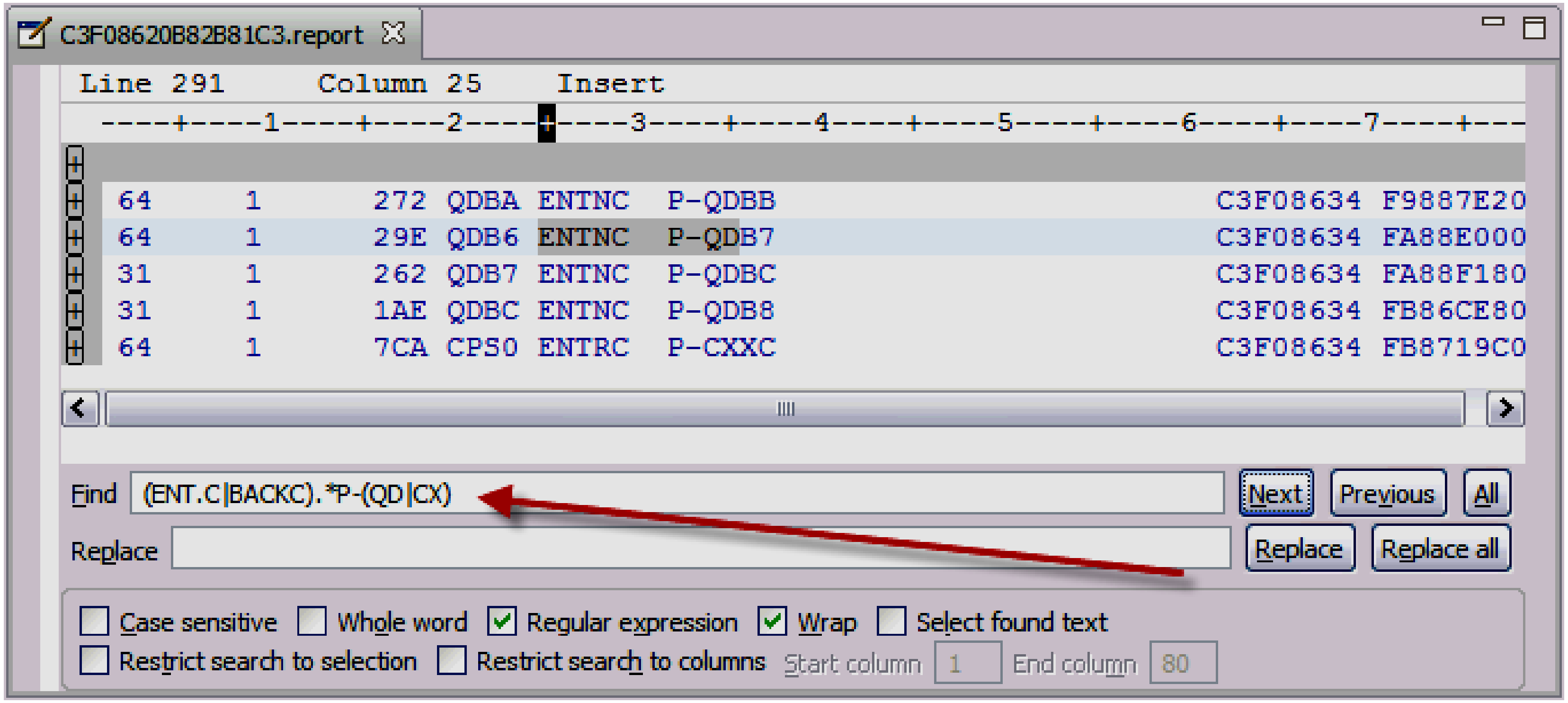
- **The Files subsystem is essentially a GUI FTP client. Double clicking the file will open the file in LPEX.**



Remote Systems
Team
New Connection
Local
LINUXTPF
TPFMVSXA
9.57.13.89
9.57.13.73
Debug
qdb0 tbp
ECB Monitor
Performance Analyzer
SOAP Message Handlers
TPF Dump Viewer
Provider Web Services
Consumer Web Services
Files
My Home
bin
dev
etc
IBMsyserr
proc
sys
tmp
C3F08620B82B81C3.report
ftp.log011201

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*)**



C3F08620B82B81C3.report

```
Line 291      Column 25     Insert
 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

 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

z/TPF Debugger Deliverable Details

Description	z/TPF APAR	TPF Toolkit Level	TPFUG Requirement
ALASC View DETAC View Watchpoint enhancements	PJ36136 PUT6	V3.4.4	V08009S V08011S V08018S/V08008S
ECB Launcher Command TPFMEmfill	PJ36136 PJ36686 PUT6	V3.4.4	V08007S 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	PJ35430 PUT6	V3.4.2	V08061S V08029S
Malloc View Register by User Defined (transaction trapping) Trace Log Enhancement	PJ36059 PUT6	V3.4.3	V08036F V08031S V07008F V08001S V08008S

z/TPF Debugger Deliverable Details

Description	**z/TPF APAR**	**TPF Toolkit Level**	**TPFUG Requirement**
Malloc View	PJ36059 PUT6	V3.4.3	V08036F V08031S
Register by User Defined (transaction trapping)			V07008F V08001S
Trace Log Enhancement			V08008S

Appendix A - Debug Server Access Control

- **ZDBUG ACCESS does not replace the CDBPUX user exit (defined in cdbpux.c)**
- **CDBPUX is implemented by your administrator. It provides complete flexibility, especially when used with ZDBUG ACCESS**
- **The return values for this user exit have changed from TRUE/FALSE to:**
  - **CDBPUX_NO_ACCESS**: request is rejected regardless of access rules defined by ZDBUG ACCESS command
  - **CDBPUX_FOLLOW_RULES**: request is processed according to the access rules defined by ZDBUG ACCESS command
  - **CDBPUX_FULL_ACCESS**: request is granted with full access regardless of access rules defined by ZDBUG ACCESS command

Appendix A - Debug Server Access Control

- **Example for Access Control with CDBPUX**
  - CDBPUX may be used to follow or override access control in your z/TPF system based on the conditions that you define
  - Here is an example of limiting or supplying debugger privileges to a specific user:

```
if((packet->operation == UD_CLIENT_OP_DEBUG) &&
(strcmp(packet->workstationName,“jsmith”) == 0))
```

if both of these condition are met and…

CDBPUX returns…	and Access Control for any debug request is set to…	the request is…
CDBPUX_NO_ACCESS	NODBUG	Rejected
	DBUG	Rejected
CDBPUX_FOLLOW_RULES	NODBUG	Rejected
	DBUG	Allowed
CDBPUX_FULL_ACCESS	NODBUG	Allowed
	DBUG	Allowed

Appendix B - TPF Code Coverage

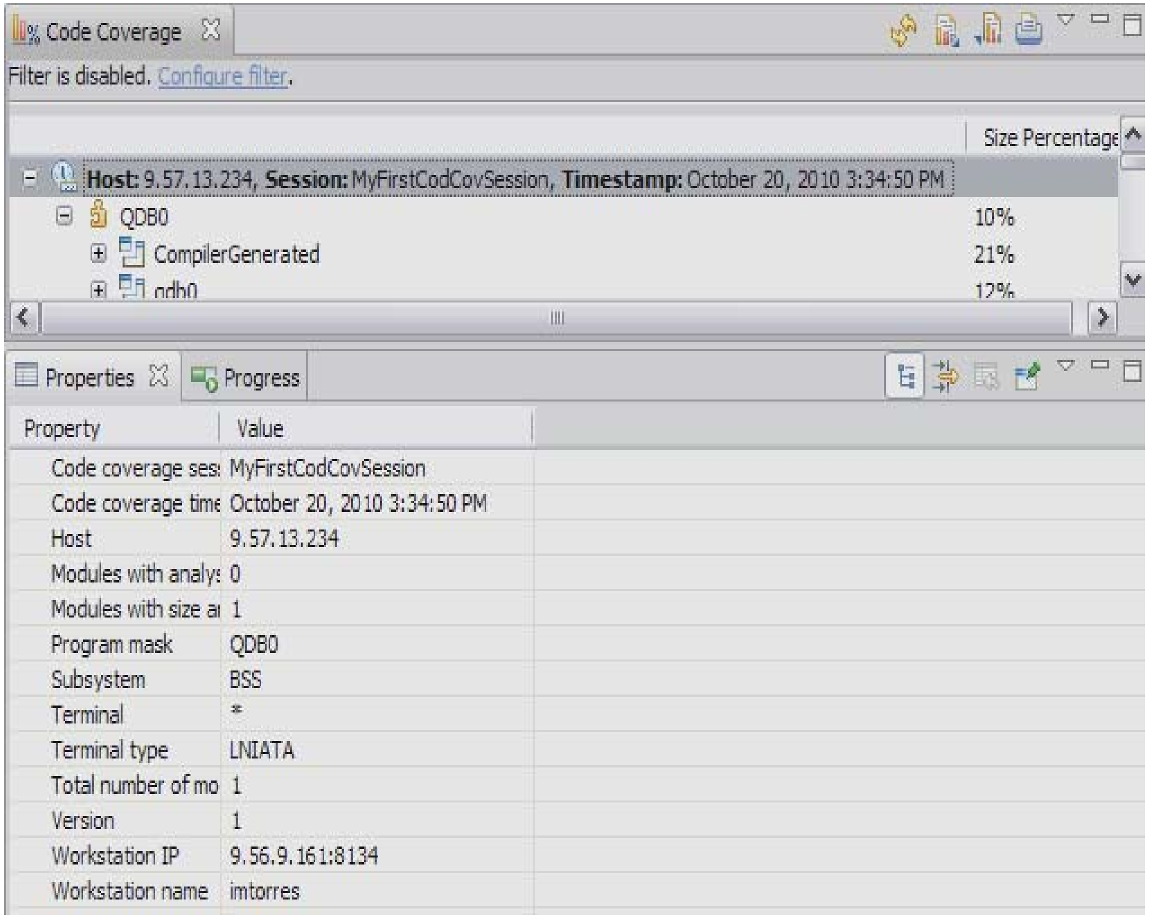
- **You can perform the following actions from the session:**
  - **Register for collection.** This step creates a code coverage registration entry on TPF. This reserves those modules for this user, so no one else can register those modules.
  - **Start collection.** This step initiates the code coverage collection on z/TPF.
  - **Save collection.** This step generates collection results that can be viewed in the code coverage view in the TPF Toolkit. Although the data is saved, the code coverage tool will continue performing collection of the code coverage data. Each time you save, you will be over-writing the data that was last saved, even if analysis was previously run.
  - **Save and Stop collection.** This step performs the Save collection action as described above. The stop action stops the collection of code coverage data.
  - **Cancel collection.** Discards all collected results. If save was previously performed, that saved data is unaffected.
  - **Cancel registration.** This step ends the code coverage registration and releases the modules that were previously reserved.
  - **Show collection status.** This will show you the state of the collection.

Appendix B - TPF Code Coverage

- **Once the timestamp entry is created, you can perform the following actions from the timestamp entry:**
  - **Perform size analysis.** This step determines percentages based on the data collected between start collection and the save (or save and stop) collection times. Size analysis will be performed on all modules in the registration entry that have been executed. If at the time the session was created, you specified “automatically perform size analysis,” z/TPF will automatically perform the size analysis when you select “save collection” or “save and stop collection.” However, the results will not appear in the Code Coverage view until you select “view results.”
  - **View results.** This step will display the results of the collection in the code coverage view. If size analysis has not been performed on this timestamp entry, the module results will display. If size analysis has been performed, then the module, object, and function results will display.

Appendix C - TPF Code Coverage

- **Properties pane at timestamp level**



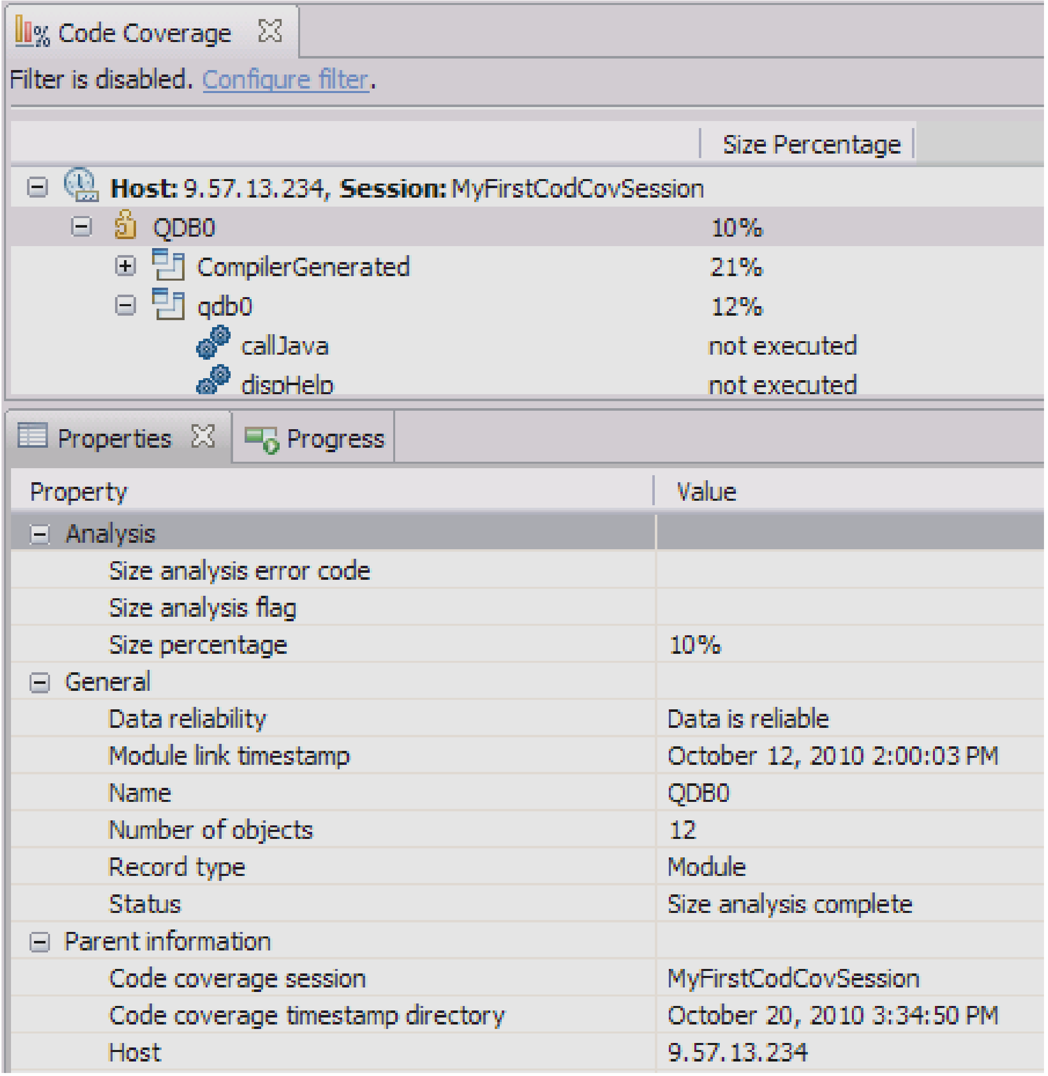
Host: 9.57.13.234, **Session:** MyFirstCodCovSession, **Timestamp:** October 20, 2010 3:34:50 PM

	Size Percentage
QDB0	10%
CompilerGenerated	21%
qdb0	12%

Property	Value
Code coverage ses:	MyFirstCodCovSession
Code coverage time	October 20, 2010 3:34:50 PM
Host	9.57.13.234
Modules with analy:	0
Modules with size ai	1
Program mask	QDB0
Subsystem	BSS
Terminal	*
Terminal type	LNIATA
Total number of mo	1
Version	1
Workstation IP	9.56.9.161:8134
Workstation name	imtorres

Appendix C - TPF Code Coverage

- **Properties pane at module level**



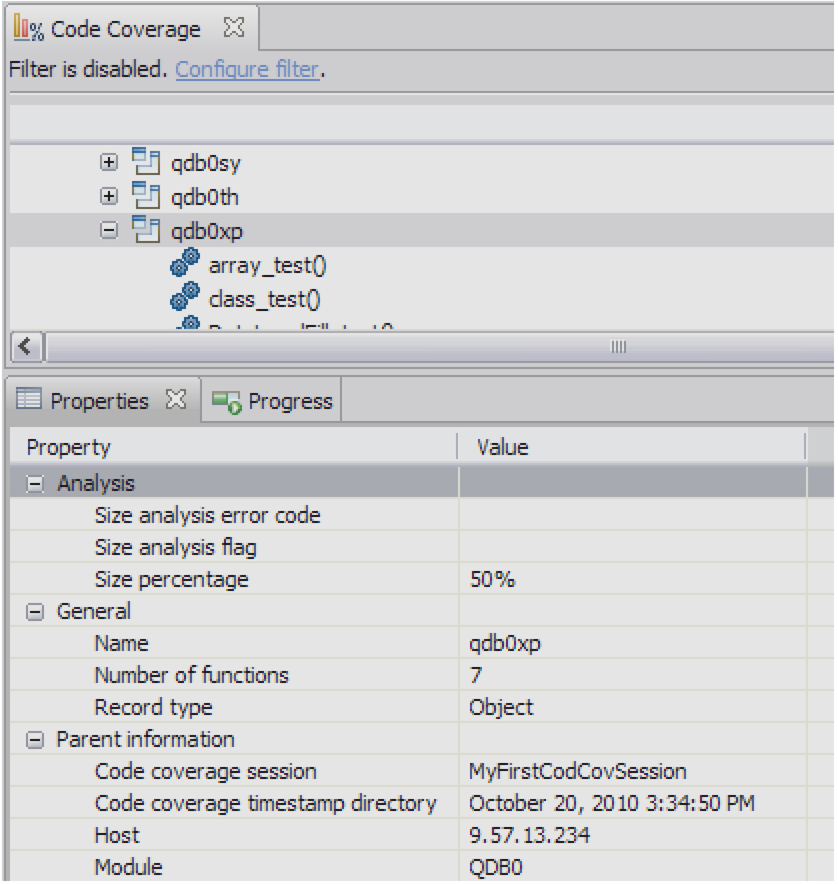
Filter is disabled. Configure filter.

	Size Percentage
Host: 9.57.13.234, **Session:** MyFirstCodCovSession	
QDB0	10%
CompilerGenerated	21%
qdb0	12%
callJava	not executed
dispHelp	not executed

Property	Value
Analysis	
Size analysis error code	
Size analysis flag	
Size percentage	10%
General	
Data reliability	Data is reliable
Module link timestamp	October 12, 2010 2:00:03 PM
Name	QDB0
Number of objects	12
Record type	Module
Status	Size analysis complete
Parent information	
Code coverage session	MyFirstCodCovSession
Code coverage timestamp directory	October 20, 2010 3:34:50 PM
Host	9.57.13.234

Appendix C - TPF Code Coverage

- **Properties pane at object level**



Code Coverage

Filter is disabled. Configure filter.

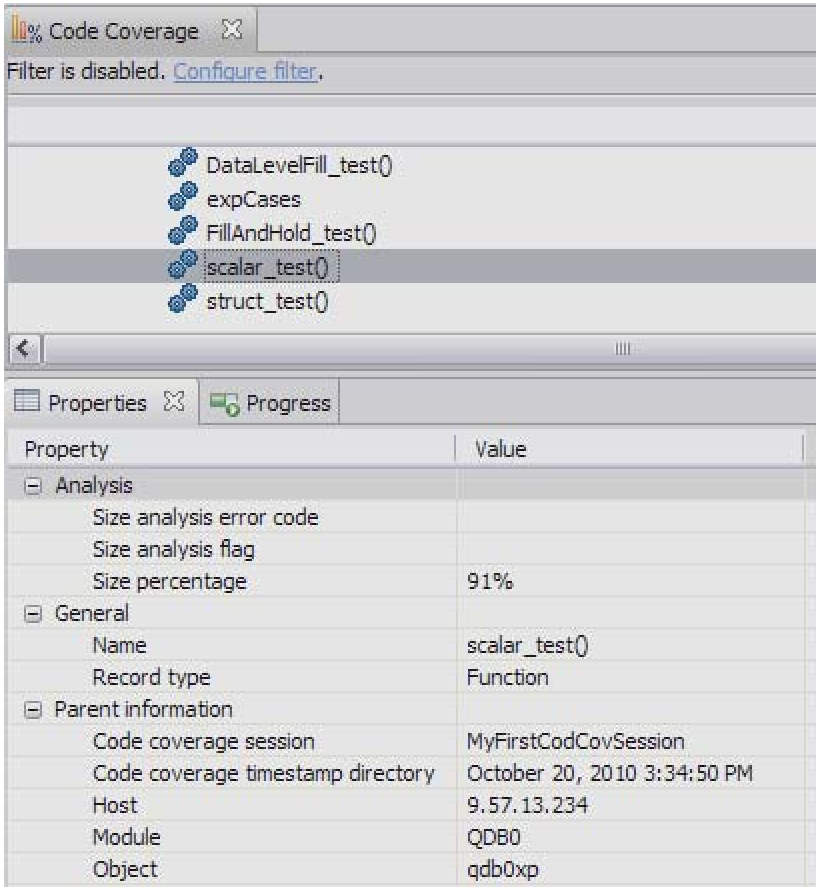
qdb0sy
qdb0th
qdb0xp
array_test()
class_test()

Properties | Progress

Property	Value
Analysis	
Size analysis error code	
Size analysis flag	
Size percentage	50%
General	
Name	qdb0xp
Number of functions	7
Record type	Object
Parent information	
Code coverage session	MyFirstCodCovSession
Code coverage timestamp directory	October 20, 2010 3:34:50 PM
Host	9.57.13.234
Module	QDB0

Appendix C - TPF Code Coverage

- **Properties pane at function level**



Filter is disabled. Configure filter.

- DataLevelFill_test()
- expCases
- FillAndHold_test()
- scalar_test()
- struct_test()

Property	Value
⊟ Analysis	
Size analysis error code	
Size analysis flag	
Size percentage	91%
⊟ General	
Name	scalar_test()
Record type	Function
⊟ Parent information	
Code coverage session	MyFirstCodCovSession
Code coverage timestamp directory	October 20, 2010 3:34:50 PM
Host	9.57.13.234
Module	QDB0
Object	qdb0xp

Appendix D - XML Generator for C/C++

- **Classes and structures may be defined inside a segment and/or header files included in a segment for a particular object built with debug information**
- **Only classes, structures, and unions expand in the display tree**
- **Unions and structures definitions are identified in the following the format**

```
union optional_tag { definition } union_variable;
struct optional_type_name { definition } struct_name;
```

If *optional_tag* or *optional_type_name* are not specified, the union/structure is marked as "untagged"

Appendix E - Record Hold Table View

- **File Address. Possible values:**
  - The 8-byte hexadecimal file address.
  - If a general data set (GDS) lock is held, 'GDS' followed by the 7-byte module, cylinder, head, and record (MCHR) address is displayed.
- **SS** - Subsystem Name.
- **PGM** - Program that requested the hold.
- **Status** - Lock ownership status of the current ECB. Possible values:
  - Waiting - This ECB is waiting for the lock. This value alone does not correspond to WAITER status in the ZDRHT display.
  - Holding - This ECB is the holder of this lock in this processor and in the external lock facility (XLF). This value corresponds to COMPLETE status in the ZDRHT display.
  - CommitScopeHeld - This value means that an unhold has been issued but a txcommit or txrollback has not yet occurred.
- **Holder** – Show which ECB is holding the lock. Possible values are: 4-byte address of the ECB, Other Proc, or Self
- **Queue Position** - The position of the current ECB in the waiter list for this processor

Appendix F - TPF Code Coverage

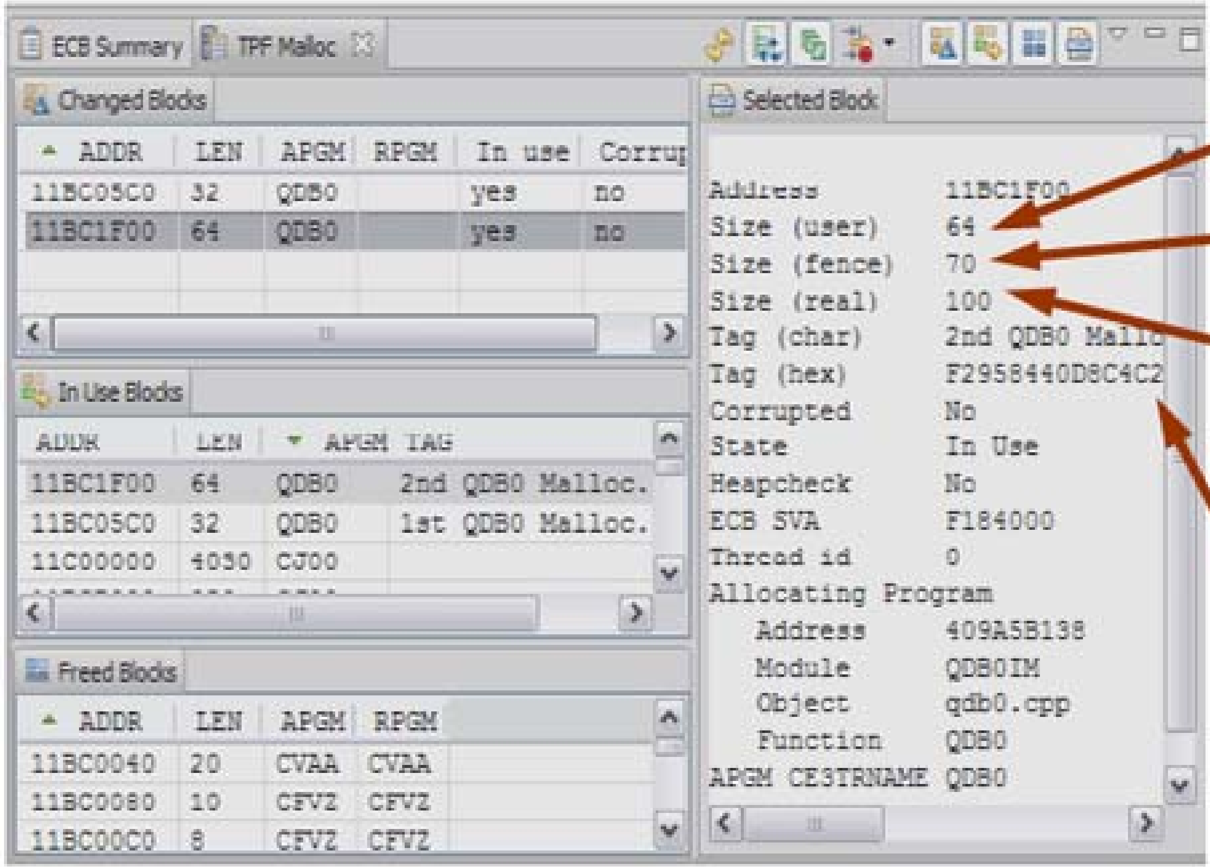
- **All directories and files are stored in the BSS file system on TPF under:**

  ```
  /rootDirectory/workstation.sessionName/timestamp
  Example:
  /TPFCodCov/imtorres.MyFirstCodCovSession/20101020033450
  ```

- **Each timestamp directory contains**
  - *results.ccvs* - The results of the collection are written out into the results.ccvs file. This is the file that the TPF Toolkit uses to show the user the percentages.
  - *annotations.txt* - A free form text file that is associated with a particular timestamp entry.
  - *results.ccvb* - The raw collected data.

Appendix G - Malloc View Filters

- **This enhancement also changed the display for the selected blocks**



Changed Blocks

ADDR	LEN	APGM	RPGM	In use	Corrup
11BC05C0	32	QDB0		yes	no
11BC1F00	64	QDB0		yes	no

In Use Blocks

ADDR	LEN	APGM	TAG
11BC1F00	64	QDB0	2nd QDB0 Malloc.
11BC05C0	32	QDB0	1st QDB0 Malloc.
11C00000	4030	CJ00	

Freed Blocks

ADDR	LEN	APGM	RPGM
11BC0040	20	CVAA	CVAA
11BC0080	10	CFVZ	CFVZ
11BC00C0	8	CFVZ	CFVZ

Selected Block

```
Address           11BC1F00
Size (user)       64
Size (fence)      70
Size (real)       100
Tag (char)        2nd QDB0 Mallo
Tag (hex)         F2958440D8C4C2
Corrupted         No
State             In Use
Heapcheck         No
ECB SVA           F184000
Thread id         0
Allocating Program
    Address       409A5B138
    Module        QDB0IM
    Object        qdb0.cpp
    Function      QDB0
APGM CE3TRNAME    QDB0
```

Size (in hex):

user: number of bytes requested by user in malloc instruction

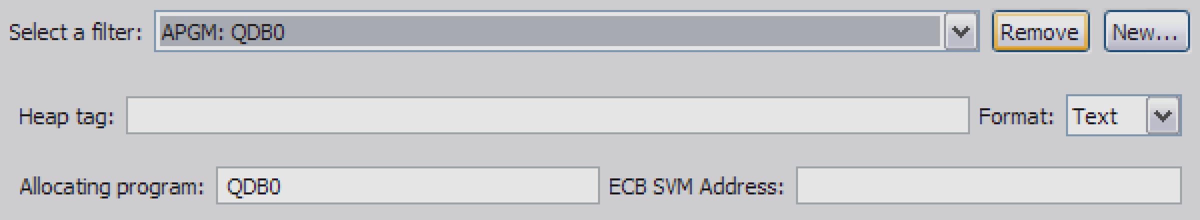
fence: user size + size of malloc fence (0xFF’s)

real: actual block allocated by system, ending in doubleword boundary

EHEAPC tag value in char and hex representation

Appendix G - Malloc View Filters

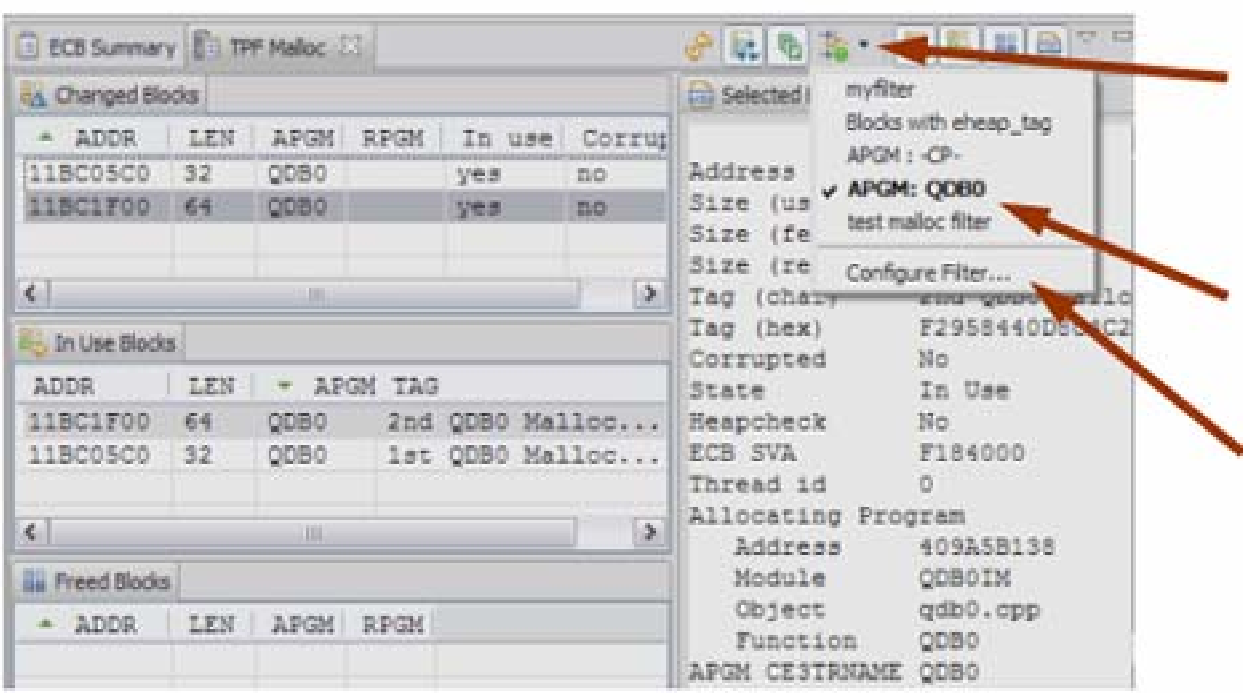
- **Example:**
  - Filter by allocating program name QDB0



Select a filter: APGM: QDB0 | Remove | New...

Heap tag: | Format: Text

Allocating program: QDB0 | ECB SVM Address:



Changed Blocks

ADDR	LEN	APGM	RPGM	In use	Corrup
11BC05C0	32	QDB0		yes	no
11BC1F00	64	QDB0		yes	no

In Use Blocks

ADDR	LEN	APGM	TAG
11BC1F00	64	QDB0	2nd QDB0 Malloc...
11BC05C0	32	QDB0	1st QDB0 Malloc...

Freed Blocks

ADDR	LEN	APGM	RPGM

```
Address
Size (us
Size (fe
Size (re
Tag (char)
Tag (hex)          F2958440D...C2
Corrupted          No
State              In Use
Heapcheck          No
ECB SVA            F184000
Thread id          0
Allocating Program
   Address         409A5B138
   Module          QDB0IM
   Object          qdb0.cpp
   Function        QDB0
APGM CE3TRNAME     QDB0
```

Filter is enabled
(toggle button to disable)

Filter in use

Create/update filters

Trademarks

- IBM and TPF Toolkit V3.4 are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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.