This user’s guide is in draft form, provided for your feedback, on an as-is basis without warranty of any kind. You grant IBM the right to use any data, suggestions, or written materials that are related to the user’s guide and that you provide to IBM. Concerning any ideas, know-how, concepts, techniques, inventions, discoveries or improvements related to the user’s guide and that you provide to IBM, you grant to IBM a non-exclusive, irrevocable, unrestricted, worldwide and paid-up right and license to include the foregoing in any product or service, and to use, manufacture and market any such product or service, and to allow others to do any of the foregoing.
Note: Before using this information and the product it supports, be sure to read the general information under “Notices” on page 56.
# Table of Contents

**Preface** ..................................................................................................................6

**Eclipse Basics** ........................................................................................................7

**Features** ..................................................................................................................8

- Remote System Explorer (RSE) Perspective..........................................................8
- Remote Systems .......................................................................................................8
- Debug Subsystem ....................................................................................................8
- ECB Monitor Subsystem (z/TPF only)......................................................................8
- TPF Dump Viewer Subsystem (z/TPF only).............................................................10

**Debug Perspective** ...............................................................................................11

- Breakpoints View ..................................................................................................11
- Data Level View .....................................................................................................14
- Debug Console View .............................................................................................15
- Debug View ...........................................................................................................16
- DECB View ............................................................................................................18
- ECB View ...............................................................................................................19
- Editor View .............................................................................................................20
- Memory View ..........................................................................................................22
- Modules View ........................................................................................................23
- Monitors View ........................................................................................................23
- Registers View .......................................................................................................24
- SW00SR View .......................................................................................................24
- Variables View .......................................................................................................26

**Understanding** ....................................................................................................27

- CTEST ....................................................................................................................27
- Debug Information ................................................................................................27
- Debug Session Profiles – Program profiles.........................................................29
- Debugging Optimized code on z/TPF.................................................................29
- Global Symbols.....................................................................................................30
- Heap Check Mode (z/TPF only)...........................................................................31
- Running debugger in 1052 state (z/TPF only)......................................................31
- TPFSYM ................................................................................................................31
- TPFXMLGEN .........................................................................................................31
- UCST ......................................................................................................................32

**How do I...** .........................................................................................................33

- Registering............................................................................................................33
  - Q1: ...set up a connection for registration?.......................................................33
  - Q2: ...create debug registration entries to debug my applications?................33
- Starting the Debugger............................................................................................34
Q1: ...get my TPF IP address? ................................................................. 34
Q2: ...set up my workstation IP address correctly? ................................. 35

Debugging Program .................................................................................. 35
Q1: ...debug only the module I'm interested in? ....................................... 35
Q2: ...use debug filters to limit tracing to specific programs and display the
debug filters I have in place? ................................................................. 35
Q3: ...include ADATA in a loadset? ......................................................... 35
Q4: ...know the correct debug information is loaded? .............................. 35
Q5: ...look at the ECB contents? ............................................................ 40
Q6: ...monitor #define macros? .............................................................. 40
Q7: ...monitor an expression? ................................................................. 40
Q8: ...monitor a global expressions? ....................................................... 41
Q9: ...trace production code? ................................................................. 41
Q10: ...trace a macro or a macro group? ............................................... 41
Q11: ... use debugger features to help me debug a dump? ........................ 41
Q12: ...change the time the debugger waits for when I get the message “Do
you wish to wait for the debug engine to respond”? ............................ 44
Q13: ...enable/disable hovering in the editor view? ................................. 44
Q14: ...see my ECB trace for the ECB I’m debugging? (z/TPF only) ........... 44
Q15: ...work with source from another users HFS? ................................. 44
Q16: ...see my variables in a different format? ........................................ 44
Q17: ...see what my pointer points to? .................................................. 44
Q18: ...set up my source lookup path? .................................................... 45
Q19: ...override my source file in my editor view? ................................. 45

Using the Debug Console ......................................................................... 45
Q1: ...issue a GETCC, RELCC, and etc on data levels? ............................. 45

ECB Monitoring ....................................................................................... 45
Q1: ...use an ECB monitor? ................................................................. 45
Q2: ...save an ECB monitor session for later view? ............................... 45

Dump Viewing ........................................................................................... 45
Q1: ...capture a dump? ........................................................................ 45
Q2: ...manage dumps on the TPF System? .......................................... 45
Q3: ...use the dump viewer? ................................................................. 45

Reporting an Error .................................................................................. 46
Q1: ...collect an EPDC Trace? ............................................................... 46
Q2: ...collect the TPF Toolkit .log (error log) file? ............................... 46
Q3: ...collect a TPF debugger trace (TPFDbgtrace)? ............................ 46

Frequently Asked Questions (FAQs) ....................................................... 47

Registering ............................................................................................... 47
Q1: Why is the debugger not registering my program? ............................ 47
Q2: Why does the debugger stop in a program other than the programs registered in the program mask? ................................. 47

Starting Debugger .................................................................................................................................................... 47

Q1: Why is the debugger not finding my assembler code? ................................................................. 47
Q2: Why is the debugger finding the wrong code? ............................................................... 47
Q3: Why is the debugger not finding my C/C++ code? ............................................................... 47
Q4: Why does the debugger issue a 0ADB01 dump when I tried to initiate a debugger session? ............................................................................. 48
Q5: Why is the debugger not trapping my ECB? ............................................................................. 48

Debugging Program ............................................................................................................................................... 49

Q1: What is the significance of blue and green font color in my source? .............................. 49
Q2: Why is my code not executing sequentially in the debugger? ........................................ 49
Q3: Why are the execute buttons disabled when debugging my program? ........................ 49
Q4: Why can’t I step into a function? ............................................................................................... 49
Q5: Why is the debugger saying my program is not in the stepdebug list? .......................... 49
Q6: Why doesn’t the debugger save my source lookup path? ............................................. 49
Q7: Why is TPF Toolkit slow at locating source files? .................................................. 50
Q8: Why doesn’t the debugger save my breakpoints? ........................................................... 50
Q9: What does the message “Execution may have stopped between statements” mean? ........................................................................................................ 50
Q10: Why do I get an error when I enter a TPF command on the Debug Console? .......................................................................................................................... 51
Q11: What does the message "Time Out Occurred: Application may be in an infinite loop. Execution may be between statements." mean? ............................................................................. 51

Dump Viewing ..................................................................................................................................................... 52

Q1: Why can’t I see my dump in the dump viewer? ............................................................................. 52

Reporting error ................................................................................................................................................... 52

Q1: What information will IBM require to diagnose the problem? .................................. 52

Index ................................................................................................................................................................. 54

Notices ............................................................................................................................................................... 56

Trademarks .......................................................................................................................................................... 56
Preface

This document is intended for use with TPF 4.1 and z/TPF platforms. The term “TPF” is used in this document to label features, scenarios, and etc applicable to both TPF 4.1 and z/TPF. Therefore, the reader is advised to assume the topics covered in this guide are for both TPF versions unless otherwise specified.

The purpose of this document is to solicit input from TPF debugger users regarding documentation for TPF Debugger in both the TPF Toolkit and TPF Information Center.

In order to provide the reader with further information about a topic, the document will reference both the TPF Toolkit and the TPF Information Center using keywords. The keywords are listed as a set of terms within quotes and divided by spaces. Keywords are catalogued under Toolkit keywords and TPFIC keywords, where “Toolkit” suggests entering the keywords in the TPF Toolkit Help and “TPFIC” suggest entering the keywords in the TPF Information Center. However, all Toolkit keywords suggested throughout the document are based on the TPF Toolkit 3.2.5 help documents and have not been tested for accuracy on back level versions.
Eclipse Basics

Eclipse is an open source framework that serves as the foundation of the TPF Toolkit. Eclipse provides a great deal of flexibility in terms of Perspectives, Views, Wizards, Preference pages, development environments and etc.

- Context sensitive help is available by hovering the cursor over a label, text box, or etc and simultaneously pressing the F1 key. A window will open to give more information regarding the object that was hovered over.

*Toolkit keywords: “workbench fundamentals”*

Toolkit help contents pane:

- “Developing in the IDE->Workbench User Guide”
- “Developing in the IDE->Workbench User Guide->getting started->Basic Tutorial”
- “Developing in the IDE->Workbench User Guide->Reference->User Interface Information”
Features

Remote System Explorer (RSE) Perspective

TPFIC keywords: “Debug Server” “ZDBBG” “ZDEBUG” “ZINET”


The remote system explorer (RSE) perspective provides a set of views that facilitate the process of connecting to remote systems. User must create a connection to the TPF system from the TPF Toolkit RSE perspective before using the debugger services provided by TPF system through the RSE subsystems. The following RSE subsystems allow users to examine ECBs through the Debug Perspective:

- DEBUG: to debug a live ECB
- ECB Monitor: to view a snapshot of a long running ECB with the debugger
- TPF Dump Viewer: to view an ECB dump with the debugger.

The Debug Server on the TPF system must be active before the RSE can connect to the TPF system.

• Remote Systems

This view allows the user to create remote connections to different platforms, monitoring ECBs and dump capturing/viewing.

- Create a TPF connection. Toolkit keywords: “Creating an RSE connection to a TPF system”

• Debug Subsystem

The Debug Subsystem is used to define and manage debug registration sessions (otherwise called debug registration entries) to use in order to start the TPF Debugger for examining an ECB in a TPF system.

- Create a new TPF debugger session. Toolkit keywords: “Creating a new debug registration session” “TPF terminal protocols”

• ECB Monitor Subsystem (z/TPF only)

TPFIC keywords: “ECB monitor” “ZDECB”

Toolkit keywords: “ECB monitor”
The ECB Monitor subsystem is used to display in-use ECBs currently on the z/TPF host. Each ECB is represented by its SVA address. This subsystem is meant to be used with long running ECBs. Also, the ECB Monitor subsystem can be used to perform a ‘snapshot’ debug of any ECB in the list.

- Add Filters to the ECB Monitor
  - Filters are used to specify what ECBs are displayed. This filter will display ECBs older than or equal to the number of seconds specified.
  - To create a new ECB Filter, right click on the ECB Monitor subsystem and specify the filter. The minimum is 5 seconds.
  - The default filter lists all ECBs on the system that are older than 5 seconds. The ECB list is similar to specifying ZDECB 5 on the z/TPF host.

- Set ECB Monitor Properties
  - From the ECB Monitor subsystem, right click and select Change properties. The following properties can be altered:
    - Under Subsystem, select the number of the port on which the z/TPF server is listening
    - Under ECB Monitor Query, specify the number of seconds the ECBs should be older than.

- Access ECB lists on the z/TPF host
  - When working with this list, it is important to refresh the view so as to get the most current list of ECBs running. To do this, right click on the ECB Filter and select “Refresh.” *Toolkit Keywords: “Refreshing a Filter”*
  - Right click and select “Show in Table” to view the list of ECBs as a table in the Remote System Details view.
  - Details for each ECB can also be viewed in the Properties View. Clicking on a particular ECB address will bring up its details in the Properties View.
    - Change the filter string to specify different ECBs to search for.

- Obtain a snapshot of an in-use ECB
Before taking a snapshot of the ECB, it is important to refresh the view to ensure that the ECBs listed are current.

To obtain the snapshot, right click on the ECB.

Any errors that occur while trying to get the snapshot will appear in the TPF Toolkit Console View.

This snapshot is static data. Execute type commands (resume, step into, etc) cannot be issued from this debug session. The snapshot can be saved by using the **ECBSnapshot** in the Debug Console View.

**TPF Dump Viewer Subsystem (z/TPF only)**

*TPFIC keywords: "viewing dumps" "ZDDMP"*

*Toolkit keywords: “dump viewer” “dump” “debugging dump”*

The dump viewer allows the user to view a dump that has been captured in the debug perspective. The dump is stored on the file system.

- **Dump Capturing**
  1. Make sure that the debugger server is started by using ZINET START S-DBG
  2. Make sure dump information goes to the debugger by using ZASER DBG. Dumps suppressed by DUMPOFF or NODUPL ZASER options will not be captured.
  3. Initiate the process that will cause the dump to occur.
  4. The TPF console will indicate that a dump control record has been added to the processor.

- The dump viewer subsystem is the filter manager for all dumps. Right click action on the dump viewer subsystem allows the user to:
  - Create a new filter
  - Show all filters with additional information in table format
  - Refresh all dump filters.

- **Dump View Filter**

  *Toolkit keyword: “dump viewer filter”*
• The dump viewer filter lists all dumps captured on a z/TPF system that satisfy the filter requirements.
• A dump viewer filter can filter based on program name, subsystem name, terminal address, etc.
• Right click actions on the dump viewer filter allows the user to:
  • Refresh the filter in the event of another dump being captured on the system
  • Show all dumps with additional information in table format.
  o Right click action on the selected dump allows the user to:
    Toolkit keyword: “list of dumps”
    • Debug – view various parts of memory, register, stack and variables from debug perspective
    • Detail – view general dump information
    • Delete – deletes the dump from the dump viewer database and z/TPF file system.
  o The ZDDMP command can be used to manage dumps on the z/TPF system.

**Debug Perspective**

The debug perspective provides a set of views that facilitate the process of examining an ECB. When a view from this list is not displayed in the Debug Perspective, the data contained in the view will not be shown or refreshed thereby improving debugger response time. In order to open a view, go to the menu bar and select Window>Show View>Others... The following view descriptions are in alphabetical order:

• **Breakpoints View**

  Breakpoints help control the flow of the program the user is debugging. They indicate to the debugger when execution should be halted.

  Toolkit keywords: “breakpoints view”
  o Shows to the user all breakpoints that have been set
  o Allows the user to set breakpoints
“Show breakpoints supported by selected target” button shows only the breakpoints for the current debugger session. *Toolkit keywords: “show supported breakpoints”*

“Remove all breakpoints” and “Remove selected breakpoint” buttons remove the specified breakpoints. *Toolkit keywords: “removing breakpoints”*

Toggle check button enables and disables breakpoints

“Skip all breakpoints” Toggle button allows user to temporarily skip all breakpoints. *Toolkit keywords: “skip all breakpoints”*

“Manage Compiled Language event breakpoints” button allows the user to select which event breakpoints can occur, such as stopping on system errors like SERRC and SNAPC dumps or exceptions.

- Users can select from the following options:
  - On TPF 4.1
    - TEST(ALL) – trap all ECB dumps in this debugger session.
    - TEST(NONE) – will not stop on any system errors.
  - On z/TPF
    - CAUGHT EXCEPTION – to catch all C++ exceptions when they are ‘thrown’.
    - UNCAUGHT EXCEPTION – to trap uncaught C++ exceptions when the exception has no ‘catch’ clause to catch the exception.
    - SYSTEM ERROR – trap all ECB dumps in this debugger session.

The default setting is ‘SYSTEM ERROR’ and ‘UNCAUGHT EXCEPTION’.

Once the debugger stops due to the system error or exception event, the debugger will prompt the user to choose from one of the following actions:

- Step into handler – steps into the code in the “catch” clause for that particular type of exception
- Run – runs the program
Examine – gives user the opportunity to look at the ECB state. In z/TPF, the user can issue the XCPTRAP command in the debug console to add specific types of exceptions to be trapped by the debugger to a list. Once the XCPTRAP SET command is entered, only the exceptions specified in this list will be trapped.

**TPFIC keywords:** “Event Breakpoints” “XCPTRAP”

**Toolkit keywords:** “Event Breakpoints”

Right-click actions allow the user to do the following:

- Add breakpoints. The user can add the following types of breakpoints with frequency and/or expression conditions. **Toolkit keywords:** “Using breakpoints”
  - Set address breakpoint on an address or label to stop execution at a particular location.
  - Set entry breakpoint on a function name to stop execution when a function is called.
  - Set line breakpoint on a line in an object to stop execution at a particular location.
  - Set load breakpoint on a 4-character module name to stop execution when a module is entered.
  - Set watch breakpoint on an address or expression to stop execution when the contents at a location have changed.

- Stop at all function entries causes the debugger to stop the application at the beginning of every function that is called (on TPF 4.1 LLM functions may or may not cause a stopping condition).

- Disable breakpoint prevents the debugger from stopping the application for the selected breakpoints. **Toolkit keywords:** “Enabling and disabling breakpoints”

- Enable breakpoints allows the debugger to stop the application for the selected breakpoints. **Toolkit keywords:** “Enabling and disabling breakpoints”
- Edit breakpoints allows the user to change the parameters of the breakpoint.
- Remove breakpoint deletes the selected breakpoint from the debugger session. *Toolkit keywords: “Removing breakpoints”*
- Remove all breakpoints deletes all breakpoints from the debugger session. *Toolkit keywords: “Removing breakpoints”*
- Setting a macro breakpoint on a TPF or user SVC macro allows the debugger to stop the execution of the application at the invocation of the specified macro. To create a macro breakpoint, add an **entry breakpoint**, choose to mark the breakpoint as deferred, and enter the macro name as the function name. Once created, the breakpoint view will indicate the breakpoint is a macro breakpoint. *TPFIC keywords: “macro breakpoint”*
- Setting a macro breakpoint on a macro group for a TPF or user macro group allows the debugger to stop the execution of the application at the invocation of any macro in the macro group. To create a macro group breakpoint, add an **entry breakpoint**, choose to mark the breakpoint as deferred, and enter the macro group name as the function name. Once created, the breakpoint view will indicate the breakpoint is a macro group breakpoint. The user may also create a macro group. *TPFIC keywords: “macro breakpoint” “CREGPC”*
- When double-clicking on a breakpoint, the editor view shows the location in the file where the selected breakpoint is set.

**Data Level View**

*Toolkit keywords: “data level view”*

The data level view shows the information for the 16 data levels for the selected ECB.

- In addition to the general information for each data level, the view also includes:
  - Rec Held: Record hold field indicating that the file address is held in the record hold table for the debugged ECB.
- An XML file backs the displayed fields such that the field order, field names, and which fields are displayed can be changed.
o Right-click action allows the user to do the following:
  ▪ Edit the individual fields, the “rec held” field cannot be edited.
  ▪ “Go to address” will show memory contents for the data that was selected on in the memory view.
  ▪ Edit the eb0eb.xml map file that describes the data location.
  ▪ Edit the DataLevelDisplay.xml map file that describes the layout.
  ▪ Refresh the view with any XML file changes.

o Double click action on the individual fields allows the user to edit the field.

- **Debug Console View**

  This view allows the user to issue TPF commands or debugger commands. *Toolkit & TPFIC keywords: “Debug console”*

  o The user can issue the following debugger commands in the debug console:
    ▪ HELP displays a list of all the commands, with their syntax, that may be entered by the user.
    ▪ `<command>` HELP displays more information for that particular command, for example, the objective of the command.
    ▪ Commands to perform operations on data levels:
      • GETCC assigns a storage block with the size and type specified to an ECB.
      • RELCC releases from the ECB the storage block specified by the data level and returns it to the appropriate available working storage pool.
      • DETAC detaches a storage block from the specified ECB data level while allowing the ECB data level to be reused.
      • ATTAC reattaches a working storage block to an ECB data level.
      • FLIPC interchanges the data contained in the ECB control fields associated with the 2 specified data levels.
- **STEPDebug** allows users to manage the step debug list that contains program names that step debug execution is limited to. *Toolkit keywords: “step debug”*
- **ECBTrace** displays the ECB trace information. It allows users to see functions and macros recently executed by the application. *(z/TPF only)*  
  *TPFIC keywords: “ecbtrace”*
- **ECBHeap** allows the user to view ECB heap usage statistics for the application. *(z/TPF only)*
- **ECBSnapshot** saves an ECB snapshot file and adds it to the dump control table to allow easy access from the TPF dump view. *(z/TPF only)*
- **TPFTimeout** allows the user to set a time out value for an application (not the TPF debugger) to identify an infinite loop situation.
- **TRLOG** enables/disables ECB trace (macro and function trace) logging for the debugged ECB. *(z/TPF only)*  
  *TPFIC keywords: “trlog”*
- **TPFDbgtrace** turns on/off diagnostic trace for the debugger.
- **FREE** frees LNIATA when the debug console will no longer use it.
- **XCPTRAP.**
- **MORE** retrieves additional TPF command output.
  - LNIATA is not required to issue debugger commands
  - LNIATA is required for TPF commands.
  - Installing LNIATAs for the debug console view enables debug console to accept TPF commands.  
    *TPFIC keywords: “Installing LNIATA for debug console”*

**Debug View**

*Toolkit Keywords: “Running Halting- Using the Debug View”,*

Debug view displays the ECB SVA address, execution point and stack frames of the suspended ECBs for all threads in all active debugger sessions. The user can click between debugger sessions to choose which debugger session to work with. On z/TPF, the user can click between threads to choose which thread to work with (click on the desired thread
before performing any action). On TPF, the user can click a stack frame to show the
associated source in the editor view and local variables in the variables view.

The user can perform following option from the tabs:

- **Resume**: Resume the application execution. The application may run to the next
  breakpoint set by the user or run to completion if no breakpoint is hit.
- **Terminate**: Ends the application and all threads immediately; the application does
  not run to completion. Terminating an application can produce unpredictable
  results.
- **Step into**: If the instruction is a function call or a BAS type instruction in an
  assembler program, step into traces into the called function or the target of BAS
  instruction, otherwise it steps to the next sequential instruction.
- **Step over**: Steps to the next sequential instruction but not into any function or BAS
  type calls.
- **Step Return**: Runs to the caller of the current function.
- **Step filtering/Step Debug**: When this button is toggled on, the step into button will
  behave as step debug. Step debug behaves similar to step into except that the TPF
  debugger will only stop the application in modules that are in the step debug list.
  The step debug list is set using the STEPDebug command in the debug console.
  The debugger will stop at all breakpoints and step over, run, and etc is unaffected.
- **Debug UI daemon**: Start, stop or change the port number for the TPF Toolkit
  listener. The user can also retrieve the workstation IP address that is known to the
  TPF Toolkit.

There are additional options available by righting clicking a stack frame inside of the
debug view:

- **Properties**: Shows details regarding the selected stack frame.
- **Terminate and Remove**: terminate the debug session and remove the session
  from debug view.
- **Edit Source Lookup**
- **Perform Heap Check on Stop**: tells the debugger to verify that application’s
  heap has not been corrupted after the completion of any execute request (step
into, breakpoint hit or etc). If the application’s heap has been corrupted, the
debugger will produce a popup box to notify the user. This feature may
degrad performance of the TPF Debugger.

- **Stop at all function entries.**

There are additional execute functionality available by right clicking in the editor view:

- **Jump to location**
- **Run to location**

There are additional circumstances that can stop the execution of the application:

- **Breakpoints**
- **CTEST**
- **Fork:** When a fork type condition occurs (cremc, swisc_create, etc) in the
  application and the debugger was registered with the trace created entries
  checkbox checked, the debugger will show the location in the code that did
  the fork type call and present the user with a pop-up box presenting the user
  with the following options:
  
  o **Follow parent:** The parent ECB continues executing the last execute
    request (step into, run, etc). The child ECB executes as normal without
    any TPF debugger intervention.
  
  o **Follow child:** The parent ECB continues executing the last execute
    request (step into, run, etc). A new debugger session starts for the child
    ECB resulting in two active debugger sessions, one for the parent and
    one for the child that can be used completely independent of each other.
    NOTE: The module that the child ECB enters must be included in the
    debug registration entry program mask.

- **Manage Compiled Language event breakpoints**
- **Time out condition.**

**DECB View**

*Toolkit keywords: “decb view” “working with DECB view”*

The DECB view displays the information pertaining to DECBs that are created by the
application. The DECB view is similar to the Data Level view except it uses the following map files:
- idecb.xml
- DECBDisplay.xml.

- **ECB View**

  *Toolkit keywords: “ECB view” “working with ECB view”*

  The ECB view is a memory view that displays page one of the ECB similar to how the ECB is formatted in TPF dumps. The view is broken into three separate panes:
  - ECB Tree: Tree structure that helps the user to locate the mapped memory element mapped in ECB.xml.
    - Right-click actions allow the user to:
      - Edit the individual fields
      - “Find Field” brings up dialog box with a filter and search bar to assist in finding a particular element in the ECB Tree View
      - Edit the ECB.xml map file that describes the data location
      - “Set Group” adds a specific field to a group
      - “Manage Groups” brings up a dialog box that creates a specific group and removes specific groups
      - “Show Group” displays the various fields that have been selected using the “Set Group” option.
      - Expand all of the sections of ECB fields
      - “Rebuild map” refreshes the XML file changes.
  - ECB Dump: Displays the ECB memory rendered as hex.
    - Right-click actions allows the user to:
      - Edit the individual fields
      - Edit the ECB.xml map file that describes the data location. *Toolkit keywords: “memory mapping file”*
      - “Go to address” will open the memory view on the address contained in the selected field.
      - Refresh the view with any XML file changes.
- ECB EBCDIC: Pane that attempts to map the memory bytes into text using the EBCDIC encoding.
  - Right click actions allow the user to:
    - Set renderings of memory to EBCDIC and ASCII.
- Double click a field to change the value.
- Click action allows the user to go to the selected address in all three panes
- Hide and show the three default different panes from the button options
- Add and cancel different memory renderings.

**Editor View**

In the debugger Editor View, right click for menu options pertaining to the current file being viewed.
- Find Text: Locates a text string in the file currently being viewed.
- Find Function or Entry Point: Locate a particular function or entry point in the program.
- Add Breakpoint:
  - For C/C++, line breakpoint is created at the selected location
  - For ASM, address breakpoint is created at the selected location.
- Add Watch Breakpoint
- Jump to Location: The program will jump to the current position of the cursor in the editor. Code between the current execution point and the current position of the cursor in the editor will not be executed. Use this function with care as skipping instructions can cause unexpected results for the application.
  - In TPF 4.1, the user may not be able to perform this operation in a C/C++ program if the target location is not in the same ‘block’ as the current instruction. The user can compile the program with the **NOBLOCK** option to remove this restriction.
- Run to Location: The program will run to the current position of the cursor in the editor. It will stop before the editor cursor position if the application hits an active breakpoint, an exception, or the end of the program.
Monitor memory: Highlight the variable. Right click and select Monitor Memory. Select from the menu the type of representation of the variable to display, for example, Hex and Character. The variable will now appear in the Memory View. Toolkit: “Monitoring Memory”

Monitor expression: Click on the line with the expression to monitor, then right-click and select Monitor Expression. The expression will now appear in the Monitors View. Toolkit: “Monitoring Expression”

Edit Source Lookup: Specify paths used to locate the source files. From here, the user can add or remove paths, and specify in what order the paths should be searched. To add a path, the user can choose “TPF Project”, “Remote Folders”, or “File System Directory” from the list in the “Add Source” dialog window. Toolkit: “Edit Source Lookup”

Change Text File: Change the version of the text file that is currently being displayed for the selected stack frame. This option allows the user to specify the full path name of the overriding file. This is useful when the file that is being displayed does not match the version of the file that is currently executing.

Switch view: Switch between the disassembly view and the source view. Toolkit: “Switch View” “Show Disassembly”

The source files in the Editor View can also be selected in the following ways:

- From the Debug View, double click on the stack frame of the file to open. Toolkit: “Edit Source Lookup”
- From the Modules View, double click on a file.

In addition to the menu options, the following capabilities are also available from the debugger Editor View. These Debugger Editor options can be set from Windows > Preferences > Run/Debug > Compiled Debug:

Toolkit: “Hover debugger”

- To see the current value of a variable based on the current location in the trace, hover over any instance of the variable in the editor view.
- Highlight and hover over a complex expression.
- Hovering can be enabled/disabled by the user

Miscellaneous:
To go to a particular line in the text: CTRL+L and enter the line number. When the source file cannot be located, the following options are displayed.

- Edit Source Lookup Path.
- Change Text File: Select this if the location of the file is known.
- Show Disassembly: Select this if the source is not available.

**Memory View**

*Toolkit keywords: “Monitoring Memory”*

The Memory view allows the user to view and change the contents of memory used by the application. The view is split into two different panes:

- **Monitors pane**
  - Right click actions allow the user to:
    - Add a new memory monitor from an expression or an EVA address in the format of 1000 or 0x1000 to display memory location hex 1000
    - Remove memory monitor
    - Reset the pointer to the base address of the selected memory monitor
    - Change the default rendering to ASCII or EBCDIC.
  - Memory renderings: shows the memory at the given address of the monitor
    - Right click actions allow the user to:
      - Add and remove memory renderings (XML files, ASCII, EBCDIC).
      - Reset the pointer to the base address of the selected memory monitor
      - Go to a selected address that was clicked on in the view.
      - Change the format of the view to display in different number of rows and columns
      - Show and hide address column
    - Double click action allows the user to edit the value.

Additionally, the user can perform the following options on the memory view:
Toolkit keyword: “Inspecting memory in memory view”

- Switch memory monitors
- Add a new rendering of the same memory
- Toggle split pane
- Link the memory rendering panes
- Add new instances of the memory view.

• Modules View

Toolkit keywords: “Modules view”

This view displays all modules in the application stack (and program nesting level for TPF 4.1) when the debugger session is started. The debugger adds additional modules as the debugged application is executed through additional modules. From a module in the modules view, the user can:

- Show all the objects in a module
  - Show all the files that make up an object
  - Double-clicking on a file opens the file in the editor view
    - Show all the functions in a file
    - Double-clicking on a function opens the file at that location in the editor view
    - Right-clicking on a function allows you to set an entry breakpoint.

• Monitors View

Toolkit keywords: “Adding an expression to the Monitors view”

This view displays expressions (for example, a local variable, a register, arithmetic expression, or global symbol) that have been selected for monitoring by the user.

- Double-click on the desired expression or variable in the monitors view allows the user to update the current value.
- Right-click actions allows the user to:
  - Dereference pointers
  - “Monitor memory” add the expression to the memory view
  - Change representation of the selected expression
- Change value of the selected expression
- “Monitor new expression” adds a new expression into the monitors list
- “Remove monitored expression” deletes the selected expression from the monitors list
- “Disable monitored expression” disables the selected expression in the monitors list, no further changes will be made to this variable unless enabled
- “Show type names” displays the data type for all monitored expressions
- “Select globals from list” adds to the monitors view the selected global variables.

**Registers View**

*Toolkit keywords: “registers view”*

The registers view shows the following groups of data:
- General purpose registers
- Floating point registers
- Control registers
- PSW.

General registers and floating point registers can be updated, but control registers and PSW cannot be updated.

- Right click actions allow the user to (**UCST** must be loaded in order to monitor register content):
  - Monitor the memory pointed by the register in the **memory view**
  - Monitor the content of registers in the **monitor view**.

**SW00SR View**

*Toolkit keyword: “SW00SR view” “working with SW00SR view”*

- SW00SR Summary Pane: Shows every created SW00SR for the selected ECB in the **Debug view** in a table format.
  - An XML file backs the displayed fields such that the field order, field names, and which fields are displayed can be changed.
Right click actions allow the user to do the following:

- Edit fields
- Go to a selected address that was clicked on in the view
- Edit the map file describing the data location sw00sr.xml
- Edit the map file describing the layout (SW00SRSummaryTable.xml)
- Refresh to rebuild the view with XML file changes

Double click actions allow the user to change the value, the DSECT column cannot be changed.

- Details of the selected SW00SR pane: Shows the following tabs contain additional information.
  - DBIFB Info: view information for the database interface block of the application
  - File Info: view selected fields
  - Context: view selected fields
  - Keys: view key information and disassembled keys
  - Core Block: view core block information used by selected SW00SR:
    - Core Block tab includes an LREC XML pane that uses a user defined XML file
    - XML files can be generated using TPFXMLGEN.
  - SW00SR: view the entire SW00SR.

Right click actions in the above tabs allow the user to:

- Edit fields
- Go to a selected address that was clicked on in the view
- Edit the sw00sr.xml map file that describes the data location
- Edit the map file that describes the data layout, these are different for each tab.

Double click action allows the user to change the value.
• Variables View

*Toolkit keywords: “Variables view”*

This view shows all the variables in the current debugged segment and their current value. Single-click on the value column allows the user to edit the contents for that particular variable.

- Double-click on a collapsed variable expands the attributes tree (if any)
- Right-click actions on a variable allows the user to:
  - “Monitor Local variable” adds the variable to the monitors view
  - Dereference pointers, the value of the dereferenced pointers are appended to the variables list
  - “Monitor memory” adds the variable to the memory view monitors list
  - Change representation of the selected expression
  - Change Value of the selected expression.
Understanding…

**CTEST**

*TPFIC keywords: “ctest” “ZAPAT”*

The ctest() function is an API call that allows users to stop the debugger at that particular location where the function is invoked. The debugger handles ctest() differently based on the state of the application:

- ctest() function is ignored if the debug information is not available:
  - In TPF 4.1, the program is built without debug information
  - In z/TPF, the debug information file is not loaded into the system
  - The program allocation table (PAT) entry of the program is marked as NODBUG.

- A debugger session will be initiated if the ECB has a debug registration entry assigned (for example, from a terminal that has a trace by terminal entry) and the program that contains ctest() is not marked as NODBUG in the PAT.

- ctest() will be treated as a breakpoint if the ECB is being debugged and the program issues ctest() is not marked as NODBUG in the PAT.

**Debug Information**

- What is debug information and what is it for?
  - Debug information is a collection of information generated by the compiler, assembler or etc that describes the application for a debugger to use.
  - Debug information describes variables (type, scope, location), what lines are executable, and etc. For TPF assembler programs, the debug information also includes the generated listing view.
  - For z/TPF only, using `–g3` will provide #define macro support.

- On TPF 4.1, what format does debug information exist, where does it exist, and how is it loaded?
TPF Debugger User’s Guide

- Assembler – Debug information is resides in ADATA and is created by TPFSYM. The ADATA file can be loaded along with the module by using ZTPLD or ZOLDR command. **TPFIC keyword: “loading ADATA”**
  - ADATA files can also be binary FTPed to the TPF 4.1 file system. For example, program ABCD should be binary FTPed to the location and format of /tmp/lst/ABCD.lst
  - UCST can be used to create smaller more manageable ADATA files.
  - See [How do I know the correct debug information is loaded?](#)

- C/C++ generated by the IBM LE compiler
  - Debug information is compiled into the C/C++ objects in an IBM OCO format.
  - How is C/C++ debug information loaded?
    - ZOLDR or ZTPLD as usual
  - See [How do I know the correct debug information is loaded?](#)
  - On z/TPF, what format does debug information exist, where does it exist, and how is it loaded?
    **TPFIC keywords: “Load z/TPF debugger information”**

- Assembler and C/C++ programs (called shared objects - BSO and CSO respectively) are in ELF format. There is an ELF section that contains the debug information in DWARF format. The entire shared object is copied to the z/TPF file system for the debugger to use for debug information. These files are stored in /tpfdbgelf based on the name of the file and the time stamp. For example, program ABCD may have debug information found at /tpfdbgelf/ab/abcd/20080321023401.

- Debug information files can also be binary FTPed to the z/TPF file system. For example, program ABCD may have debug information binary FTPed to the location and in the format of /tpfdbgelf/ab/abcd/ABCD.

- How is debug information loaded for assembler or C/C++?
  - Offline loads can be done directly from the TPF Toolkit. **Toolkit keywords:** “loadtpf” “Generating and loading loadsets to z/TPF for projects”
Alternatively, in the offline loader files (OLDR.load or TLDR.load) the @DEFINE must specify DEBUGFILES=YES. **TPFIC keywords: “@DEFINE”**

- Use ZOLDLDR or ZTPLD with the DEBUG parameter.
  - See How do I know the correct debug information is loaded?

- How are applications built with debug information?
  
  **Toolkit keywords: “Building TPF application debugging” “Setting remote assemble options” “Debugging a TPF application”**
  
  - The TPF 4.1 debug information build options are best obtained by using the following keywords.
    - **Toolkit keywords: “Building TPF application debugging”**
  
  - The z/TPF debug information build options can be obtained by using the following keywords.
    - **TPFIC keywords: “Building application programs debugging”**
    - **Toolkit keywords: “Building TPF application debugging”**

### Debug Session Profiles – Program profiles

**Toolkit keywords: “Setting debug preferences” “Creating new debug registration session”**

- Program profiles store debugger settings on the workstation. These program profiles are associated with the debug registration entry. The advantage is that the user can use different debug registration entries for specific test systems, test scenarios, or etc. The following settings are saved in the program profiles:
  - Created breakpoints
  - Edit Source Lookup path

- The program profiles can be deleted by using the preference pages.

### Debugging Optimized code on z/TPF

**TPFIC keywords: “optimized code debugging”**

The TPF debugger can be used to debug optimized code. By default, all programs are built with debug information (-g2) at optimization level 2 (-O2). As such, all that needs to be done to debug optimized code is to load the debug information to the file system. There are benefits and difficulties to debugging optimized code.
A benefit may be the ability to debug a problem that only occurs once the application code is built at higher optimization levels.

A benefit may be the ability to quickly diagnose and debug a problem without having to rebuild a non-optimized version to load to a test system when the production version is available to be loaded immediately.

A difficulty may be that some variables (depending upon compiler choices and optimizations) are no longer available to be monitored in the debugger. This is because the compiler may choose to only keep a value in a register as long as it is needed instead of saving that value on the stack.

A difficulty may be that stepping through the source view seems to jump around erratically (depending upon specific compiler optimizations). This can be the result of the compiler unrolling loops, interleaving instructions from a variety of lines to improve performance, and etc.

**Global Symbols**

*TPFIC keywords: “Global Symbols” “User Global Symbol Table” “User Symbol Override table”*

- What are Global Symbols?
  - Global Symbols are expressions that can be monitored through the Monitors view that are defined by the user or IBM. These symbols typically resolve data that exists outside of the application program scope. The user provides a function and a table entry in user symbol override table (USOT) or user global symbol table (UGST) to tell the debugger how to resolve these symbols for display to the user. IBM also provides a set of global symbols defined in the IBM global symbol table (CGST). For expression evaluation search order use *TPFIC keywords: “Global Symbols”*
  - The Monitors view provides a “select global list” option to show all available global symbols that can be monitored.
  - Global Symbols can be defined as 3 different types: direct (like the value of a pointer), indirect (like the value of a dereferenced pointer), and pointer list (like an array of pointers). Global Symbols can also take a parameter.
Caution must be used when writing the global symbol functions on z/TPF as these functions can be called from a debugger, dump viewer or ECB monitor session. Therefore, adhere to the guidelines found in the comments for ugst.cpp or usot.cpp.

For examples, see the TPF code: cgst.cpp, ugst.cpp and usot.cpp.

The TPF Debugger does not resolve TPF Globals, format 1 and format 2 (z/Only), unless a global symbol has been defined by the user.

**Heap Check Mode (z/TPF only)**

*TPFIC keywords: “heap check” “ZSTRC”*

Running the TPF debugger, dump viewer and ECB monitor with heap check mode ON is discouraged. Use ZSTRC ALTER to change the heap check mode option.

**Running debugger in 1052 state (z/TPF only)**

*TPFIC keywords: “debugger 1052” “Starting z/TPF debugger session” “ZPOOL 1052 UP”*

The TPF debugger, dump viewer, and ECB monitor can be run in 1052 state on z/TPF. However, pools (ZPOOL 1052), TCP/IP and the debug listener must be ready for use at 1052 state prior to using the TPF debugger.

**TPFSYM**

TPFSYM is an offline utility program that extracts debug information from a SYSADATA file (generated by HLASM) and creates an ADATA file (DWARF format) to be used by the TPF Debugger.

- To use TPFSYM, refer to *TPFIC keywords: “Debugging a high-level assembler program”*
- Using UCST creates smaller ADATA files.

**TPFXMLGEN**

This offline program is used to generate XML files that contain definitions of the DSECTs found in the debug information file (ADATA) generated by TPFSYM. The files generated by this tool can later be used for memory mapping in TPF Debugger in the *memory view*, *SW00SR view*, and *ECB view*. TPFSYM is a co-requisite of TPFXMLGEN.
UCST

TPFIC keywords: “Using common user symbols”

- What is UCST?
  - UCST is the User Common Symbol table. This table contains common symbols for assembler programs such as Register names, EB0EB symbols, and etc.

- What does UCST do for me?
  - UCST is used offline and online:
    - UCST is used offline at build time by TPFSYM to reduce the size of the ADATA files. Any symbol TPFSYM finds in UCST will not appear in the application’s ADATA file. TPFIC keywords: “Debugging a high-level assembler program”
    - UCST is used online by the TPF debugger to resolve symbols contained in UCST. UCST must be loaded to the TPF system with debug information.

- How do I build UCST?
  - UCST requires no special build procedure. Simply build it like another other application that is intended to be used by the debugger.

- How do I use UCST?
  - Under most circumstances, the system administrator should load UCST with debug information to the TPF system and the end users will transparently use UCST for assembler expressions (R0, R1, EB0EB fields, etc).

- How do I leverage UCST?
  1. Add common macros (like AAA), equates, and etc to ucst.asm.
  2. Rebuild UCST
  3. Rebuild (TPFSYM) all assembler programs against the new UCST
  4. Load the new UCST and new assembler programs.
How do I &

Registering

Q1: …set up a connection for registration?

A1: In the RSE perspective, expand the section for “New Connection.” Right click on option “TPF…” Fill out the appropriate information regarding the connection name, IP address of the TPF system and description (optional).

Q2: …create debug registration entries to debug my applications?

A1: The answer to this question is rather complex. It depends on how the application to debug will be started, what conditions exist on the system at that time, and etc. The following scenarios will address common situations and how to create a debug registration entry to correctly start the debugger. See RSE and debug subsystem for more information.

• A TPF command will be entered from Prime CRAS that enters program ABCD.
  o Create a debug registration entry where the LNIATA is 010000 and the program mask is ABCD.

• Some program will CREMC program ABCD.
  o Create a debug registration entry where the LNIATA is * (trace by program) and the program mask is ABCD.

• Program QWER will CREMC multiple ECBs in program ABCD but only the 3rd instance of ABCD should be debugged.
  o Create a debug registration entry where the LNIATA is *, the program mask is QWER and ABCD, and check the trace created entry check box. Start QWER, step into the CREMC calls and choose fork follow child. A new debugger session will be started for each child.
  o Create a debug registration entry where LNIATA is *, the program mask is ABCD, and the EB0EB condition is some unique condition to the 3rd instance of ABCD. Run QWER and the desired ABCD will start the debugger.
• Create a debug registration entry where LNIATA is * and the program mask is QWER. Start QWER, run to the location where the 3rd CREMC will occur. Create a new debug registration entry where LNIATA is * and the program mask is ABCD. Step or run past the CREMC in the QWER ECB to start a second debug registration entry for ABCD. Alternatively, modify the QWER debug registration entry to include ABCD in the program mask and trace created entries. Then step or run past the CREMC and choose fork follow child.

• Program ABCD is entered by a new ECB once every second but a specific instance is desired.
  o Create a debug registration entry where LNIATA is *, the program mask is ABCD, and a condition is specified to catch the unique ECB (EB0EB or register value).
  o Create a debug registration entry where LNIATA is *, the program mask is QZZ1, modify the ABCD code call CTEST on the given condition. Can also be used to debug a DLL.
  o Create a debug registration entry where LNIATA is *, the program mask is QZZ1, modify the ABCD code to call QZZ1 (where QZZ1 just does a return) on the given condition. When the debugger session starts in QZZ1, just step return. Can also be used to debug a DLL.

• (z/TPF only) Program ABCD will be kicked off 3 times from an MQ Queue with no time delay or distinguishing characteristics.
  o Create 3 different debug registration entries with LNIATA as *, program mask as ABCD, but each debug registration entry having a different name. Register all three debug registration entries and fire off the MQ request. Three separate debugger sessions will start, one for each debug registration entry (that is, one for each ECB).

Starting the Debugger

Q1: …get my TPF IP address?

A1: Enter “ZTTCP DISP ALL” on the TPF system.
Q2: …set up my workstation IP address correctly?

A1: If the user enters a numeric form of the IP address, the TPF Toolkit will use the IP address as specified. If IP address for the workstation changes, the user will need to re-enter the new IP address.

A2: If the user enters the workstation name for the IP address, the TPF Toolkit will resolve the workstation IP address at startup of the TPF Toolkit. If the workstation is disconnected from the network resulting in an IP address change, the TPF Toolkit will need to be restarted to pick up the new IP address.

Debugging Program

Q1: …debug only the module I’m interested in?

A1: This can be accomplished by using the step debug feature. In the Debug Console View, enter the STEPDebug command and pass the desired module name to debug as a parameter. Then toggle on the step debug button and use the step into button.

Q2: …use debug filters to limit tracing to specific programs and display the debug filters I have in place?

A1: The debug filters, also called step debug, feature is used to limit tracing to a specific list of programs. Use the STEPDebug TPF debugger command through the debug console to manage the list of programs and to view the current list. The program mask in the debug registration entry does NOT limit the tracing in any way but simply indicates when the debugger session should be started.

Q3: …include ADATA in a loadset?

A1: See How are applications built with debug information?

Q4: …know the correct debug information is loaded?

A1: See debug information for information about what debug information is, how it is correctly built, and how it is loaded.

A2: TPF 4.1 assembler ADATA:
1. Verify that the ADATA is loaded at all by using ZDPAT to verify the ADATA FA (ADATA file address) contains a valid value.

```
AAES0008I 00 ==> ZDPAT qzz4 c-c
PROGRAM QZZ4

VERSION JW
BASE PAT SLOT 079ABF60
TYPE FILE RESIDENT
LINKAGE TYPE BAL
CLASS SHARED

FILE ADDRESS FC064B65
ADDRESSING MODE 31BIT
AUTHORIZATION KEY0 MONTC RESTRICT

TEST HOOK NONE
ADATA FILE ADDR ADATA NOT LOADED
```

If the ADATA FA is N/A (not available), the ADATA file needs to be loaded. See On TPF 4.1, what format does debug information exist, where does it exist, and how is it loaded?

2. Use ZDPGM command to verify that the assembler object online matches the assembler object that should be loaded. It is possible that the loaded object and the debug information are mismatched, the object and ADATA need to be loaded. See On TPF 4.1, what format does debug information exist, where does it exist, and how is it loaded?

3. The ADATA file can be examined to verify the producer string.

- For OLD records (ZOLDR ACT modules):

```
AAES0008I 00 ==> ZDPAT qzz4 c-c
BEGIN DISPLAY OF CORE COPY

PROGRAM QZZ4

VERSION JW
BASE PAT SLOT 079ABF60
TYPE FILE RESIDENT
LINKAGE TYPE BAL
CLASS SHARED

FILE ADDRESS FC064B65
ADDRESSING MODE 31BIT
AUTHORIZATION KEY0 MONTC RESTRICT

TEST HOOK NONE
ADATA FILE ADDR F40CB179
DISPLAY OF PAT SLOTS FOR QZZ4

VV LOADSET ACT NUM STAT FILE ADDR TYPE LINK HOOKS PAT ADDR ADATA FA
-- -------- ------- ----- --------- ---- ---- ----- -------- -------- _
JW QZZJW 1 ACT F40CBBD5 FR BAL NONE 048AA018 F40CBBD1
LC BASE 0 ACT FC064B65 FR BAL NONE 079ABF60 N/A
```
The qzzjw loadsets qzz4jw was built with TPFSYM at APAR level PJ32621 on the date 20080208.

Likewise, for base programs in the APRG records (TLD or ZOLDR ACCEPT modules):

```
AAEES0008I 00 ==> ZDPAT qzz4 c-c
BEGIN DISPLAY OF CORE COPY

PROGRAM      QZZ4

VERSION      JW
BASE PAT SLOT 079ABF60
TYPE      FILE RESIDENT
LINKAGE TYPE   BAL
CLASS      SHARED

FILE ADDRESS FC064B65 _
ADDRESSING MODE 31BIT
AUTHORIZATION   KEY0 MCNTC RESTRICT

TEST HOOK      NONE
ADATA FILE ADDR F40CD179
DISPLAY OF PAT SLOTS FOR QZZ4
```

```bash
VV LOADSET ACT NUM STAT FILE ADDR TYPE LINK HOOKS PAT ADDR ADATA FA
-- ------ ------ ----- -------- -------- -------- --------
```
The qzz4jw from the base was built with TPFSYM at APAR level PJ32621 on the date 20080208.

If the time stamp in the ADATA file do not match the time stamp of the object that is loaded, the ADATA needs to be loaded. See On TPF 4.1, what format does debug information exist, where does it exist, and how is it loaded?

A3: TPF 4.1 C/C++

1. Verify that the time stamp of the C/C++ object online matches the time stamp of the C/C++ object that should be loaded.

© Copyright IBM Corporation 2008. All rights reserved.
If this time stamp does not match the time stamp of the object that was built (in HFS or PDS), the module needs to be loaded. See On TPF 4.1, what format does debug information exist, where does it exist, and how is it loaded?

2. Verify that the listing has the correct compiler options. This is done by examining the generated listing file. First, verify the time stamp in the listing file matches the time stamp of the built object (in HFS or PDS). If the time stamp does not match, the object needs to be rebuilt. Second, verify the build options are correctly set for debugging. See How are applications built with debug information?

A4: z/TPF assembler or C/C++

1. The ZDDBG command provides an option to verify debug information is loaded and readable.

```
AAES0008I 00 ==> ZDDBG display dbginfo=qdb0
CDBS0026I 11.31.06 Debug Info for program QDB0:

<table>
<thead>
<tr>
<th>VERSION</th>
<th>LOADSET</th>
<th>DEBUG</th>
<th>DEBUG.FILE</th>
<th>READABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QDB0</td>
<td>LOADTPF</td>
<td>YES</td>
<td>/tpfdbgelf/qd/qdb0/20080313161439</td>
<td>YES</td>
</tr>
<tr>
<td>QDB0.JW</td>
<td>GBASE</td>
<td>YES</td>
<td>/tpfdbgelf/qd/qdb0/20080512120353</td>
<td>YES</td>
</tr>
<tr>
<td>QDB0</td>
<td>BASE</td>
<td>YES</td>
<td>/tpfdbgelf/qd/qdb0/20080304021517</td>
<td>YES</td>
</tr>
</tbody>
</table>

END OF DISPLAY +
```

Take note of the debug file name that indicates the time stamp of the corresponding load module. If the debug file is not readable, have the administrator change the file permissions of the debug file by using ZFILE CHMOD. If the DBUG field indicates NO, have the administrator change the PAT to DBUG by using ZAPAT if appropriate. If no debug information file exist, the debug information file needs to be loaded. See On z/TPF, what format does debug information exist, where does it exist, and how is it loaded?

2. Verify that the time stamp of the object online matches the time stamp of the object that should be loaded.

```
AAES0008I 00 ==> ZDMAP QDB0 o=qdb0
DMAP0003I 11.40.34 LINK MAP DATA DISPLAY
QDB0  ACTIVE IN LOADSET LOADTPF IN SUBSYSTEM BSS
PROGRAM ADDRESS 0000000395F6F000
PROGRAM SIZE 0000B1D8
qdb0  - OBJ FILE AT ADDR 0000000395F6F21C
OBJECT FILE SIZE 00000E14
COMPILED ON 2008/03/13 AT 16.14.40
```
If this time stamp does not match the time stamp of the object that was built (in HFS), the module needs to be loaded. See On z/TPF, what format does debug information exist, where does it exist, and how is it loaded?

3. Verify that the listing has the correct assembler or compiler options. This is done by examining the generated listing file. First, verify the time stamp in the listing file matches the time stamp of the built object (in HFS). If the time stamp does not match, the object needs to be rebuilt. Second, verify the build options are correctly set for debugging. See How are applications built with debug information?

Q5: …look at the ECB contents?

A1: ECB View provides a view of the ECB typically seen in TPF dumps.

A2: In the memory view, choose the green plus and hit enter (monitor nothing), by default the memory view will open to the ECB address.

A3: In the memory view, choose the green plus, enter the global symbol “ecbptr”, and choose enter. The memory view will open to the ECB.

A4: Use A2 or A3 above. Then apply an XML file like eb0eb.xml, split the pane to see two renderings at once, or etc. Toolkit keyword: “Enterprise configuration files in TPFSHARE” “Working with memory monitors”

The TPFXMLGEN xml generator can also be used to create and XML file of customized EB0EB macros.

A5: TPF 4.1 in the Monitor view, add the global variable “ECBPTR” to see the ECB formatted as a structure.

Q6: …monitor #define macros?

TPFIC keyword: “Preprocessor macro support”

A1: z/TPF build with the –g3 option to debug #define macros.

Q7: …monitor an expression?

Expressions can be for assembler, C or C++ code for example: R1, 200(R9), MyDSECTField, MyVariable, MyClassObj, MyCStructure, A + B + C, asdf || hjkl, *this, or etc. Expressions are monitored in the Monitor view.
A1: Hover over the variable. Toolkit keywords: “allow hover evaluation”

A2: Right-click a variable or expression and choose monitor.

A3: Left-click and highlight a variable or expression, then right-click and choose monitor.

A4: Use the monitor view to enter a variable or expression. Toolkit keywords: “Adding an expression to the Monitors view”.

A5: Alternatively, use the variable view to see all local variables.

Q8: …monitor a global expressions?

A1: TPF Debugger global symbols defined in USOT or UGST can be viewed by entering the variable name in the monitor view. Alternatively, use the “select global list” from the monitor view.

A2: C/C++ file scope global variables can be viewed by entering the global variable name in the monitor view.

A3: C/C++ global variables imported from an external module cannot be monitored in the debugger.

A4: The TPF Debugger does not resolve TPF Globals, format 1 and format 2 (z/Only), unless a global symbol has been defined by the user.

Q9: …trace production code?

A1: z/TPF has support for tracing production code. See Debugging Optimized code on z/TPF.

Q10: …trace a macro or a macro group?

A1: See setting macro breakpoints for tracing SVC macros and macro groups.

A2: See debug information –g3 #define macro support.

Q11: …use debugger features to help me debug a dump?

A1: Investigating TPF Dumps and application dumping conditions is a skill based on knowledge of the application, the TPF system, and a wide variety of techniques. Also, there are an infinite number of root causes for all possible dumps on TPF. Therefore, it is the task of the user to determine the cause of a dump. The debugger provides a variety of features that can be leveraged by the user to help determine the
root cause of a dump. It is up the user to understand the features that the debugger offers and how to use those features in the widest sense to find the root cause of the dump.

The TPF debugger features have been designed to operate in the widest possible sense. However, the debugger was designed for use with application ECBs, so there are inherit limitations in the implementation (including dumping conditions originating from the control program, code marked as NODBUG in the PAT, and etc). As such, some features may not yield the desired result under all circumstances.

Here are some debugger features that can be applied to all application dumps (CTL, OPR, SERRC, and SNAPC).

- On z/TPF, the Dump Viewer can be used to collect a dump to a file that can be viewed through the debugger interface. This is a means by which to statically view the state of the application at the time of the dump.
- The TPF Debugger system error features can be used to examine a live ECB that has exhibited the dump. Simply start the debugger on an ECB that will take the dump. Disable or remove all breakpoints and choose Run from the debug view. The debugger will stop when the dump occurs. Choose examine to explore the state of the application at the time of the dump. See “Manage Compiled Language event breakpoints”
- On z/TPF, use the dump viewer or the event breakpoints to get to the location of the dump. Use ECBTRACE through the debug console to see how the application ECB got to this location.
- On z/TPF, (if possible) use the execute functionality in the debug view and breakpoints to get the application close to the location of the dump. Turn on TRLOG in the debug console. Choose Run in the debug view. After the dump occurs, post process the trace log file and examine the path that was taken to get to the point of the dump.
- Suppose the dump is the result of a bad usage of a macro. Set a macro breakpoint to stop at all instances of that macro issued by the application prior to the macro being executed.
Suppose a dump only occurs when a program is entered with a specific value in an ECB field or register. A condition can be specified at registration time to only start the debugger on an ECB meeting that condition.

The following lists mention features that are useful to investigate certain dumps. However, the features discussed here can be applied to a large variety of situations.

CTL-4, OPR-4, CTL-3 or etc.

- Suppose a dump occurred because the value of a C/C++ pointer or other memory address was assigned a bad value. A watch breakpoint could be created to stop the execution of the application whenever the memory location contents are changed.

- Suppose a particular function or assembler routine branches to 0. An address breakpoint could be set up to stop the execution of the application in the routine that is causing the bad branch.

- Suppose a function causes a write protection exception when a particular parameter is passed to it. A function breakpoint could be set with an expression specifying to only stop when the parameter is that particular value.

CTL-75 or etc.

- CTL-75 dumps indicate that heap has been corrupted and detected at the time of the free. The “perform heap check on stop” feature can be used to indicate when heap corruption has occurred. This can be used in conjunction with the “stop on all functions” or other execute functionality from the debug view and breakpoints.

- Also see the watch breakpoint example in CTL-4, OPR-4, CTL-3 section above.

CTL-10 or etc.

- CTL-10 dumps indicate that the application is potentially in an infinite loop. The debugger also attempts to detect a CTL-10 situation. The time to wait for the debugger to indicate an infinite loop occurred can be changed with the TPFTTimeout command through the debug console. To ensure that the debugger stops on timeout, determine the address boundaries of the module that causes the CTL-10 and create address breakpoints at the start and end of the module.
Q12: & change the time the debugger waits for when I get the message Do you wish to wait for the debug engine to respond?

A1: A request has been sent to the remote system, and the remote system is not responding. This usually occurs because the socket connection is no longer active. See Toolkit keywords: “Setting debug preferences – Engine response time”

Q13: …enable/disable hovering in the editor view?

Toolkit keywords: “hover debug”

A1: Go to Window->Preferences->Run/Debug->Compiled Debug -> Debugger Editor. Check/uncheck the box “Allow hover evaluation.” See Editor View for more information on hovering.

Q14: …see my ECB trace for the ECB I’m debugging? (z/TPF only)

A1: In the Debug Console view, enter ECBTRACE. See Debug Console View for more information.

Q15: …work with source from another users HFS?

Toolkit keywords: “Edit Source Lookup”

A1: In the source view, right click and select Edit Source Lookup Path, add a remote folder and use the browse button to specify the path of the file. See Edit Source Lookup in Editor View for more information.

Q16: …see my variables in a different format?

A1: Right-click on the variable and choose change representation. See variables view and monitors view.

Q17: …see what my pointer points to?

A1: Right click on the pointer variable in the variables view, or monitors view and select the option to dereference the pointer.

A2: Right click on the pointer and select the option to “Monitor Memory”

A3: Right click on a character pointer variable in the variables view or monitors view to change its representation to the desired form.
Q18: ...set up my source lookup path?

A1: In the editor view, right click and select the “Edit Source Lookup Path” option.

Q19: ...override my source file in my editor view?

A1: In the editor view, right click and select the “Change text file” option. This opens a new window that allows the user to input the direct path to the source that the debugger will use.

Using the Debug Console

Q1: ...issue a GETCC, RELCC, and etc on data levels?

A1: Use the debug console view. Type “help” on the debug engine command line for more information. Issue data level commands from the command line.

ECB Monitoring

Q1: ...use an ECB monitor?

A1: See ECB Monitor Subsystem for more information.

Q2: ...save an ECB monitor session for later view?

A1: In the Debug Console View, enter the command ECBSNAPSHOT IMPORT [TARGET-u*]. See Debug Console View for more information.

Dump Viewing

Q1: ...capture a dump?

A1: See Dump Capture

Q2: ...manage dumps on the TPF System?

Toolkit Keyword: “ZDDMP” “Managing dump files” “deleting dump”

A1: In the TPF System, enter ZDDMP DISP ALL to display all dumps.

Q3: ...use the dump viewer?

A1: Please see the section on the dump viewer subsystem for more information and common questions.
Reporting an Error

Q1: …collect an EPDC Trace?

A1: In the debug perspective, go to Window->Preferences->Run/Debug->Compiled Debug. In the Compiled Debug window, check the “Trace engine connection” and browse to the desired directory for the debugger to write out the EPDC trace file.

Q2: … collect the TPF Toolkit .log (error log) file?

Toolkit keyword: “error log” “.log file to IBM for review”

A1: The error log (.log) is a text file written into the workspace/.metadata directory.

Q3: … collect a TPF debugger trace (TPFDBgtrace)?

TPFIC keyword: “debug console view” “tpf debugger commands”

Toolkit keyword: “debug console”

A1: The TPFDBgtrace is a debugger command entered through the debug console that turns on internal debugger print statements. Enter TPFDBgtrace HELP in the debug console or see segment iudtr.h for more information.
Frequently Asked Questions (FAQs)

Registering

Q1: Why is the debugger not registering my program?
   
   A1: The TPF Debug daemon has not been started in the TPF System. See the RSE Perspective for more information.
   
   A2: Search the TPF Toolkit Help for the message ID shown in the TPF Toolkit console view.

Q2: Why does the debugger stop in a program other than the programs registered in the program mask?
   
   A1: The registered program mask indicates when the debugger should be started when the specified programs are entered. The program mask does NOT have any effect on the debugger session after the debugger session is started. To limit the debugger session to debugging a set of programs, step debug should be used.

Starting Debugger

Q1: Why is the debugger not finding my assembler code?
   
   A1: ADATA file was not included in your loadset. See debug information.

Q2: Why is the debugger finding the wrong code?
   
   A1: The source lookup path is incorrect. See edit source lookup.
   
   A2: The object that was loaded does not contain your changes.
   
   A3: The object does not match the debug information file being loaded to the system. See How do I know the correct debug information is loaded?

Q3: Why is the debugger not finding my C/C++ code?
   
   A1: The source lookup path is incorrect. See edit source lookup and How do I know the correct debug information is loaded?
Q4: Why does the debugger issue a 0ADB01 dump when I tried to initiate a debugger session?

A1: The debugger was unable to reach the workstation via the IP address saved in the debug registration entry and it takes an ADB01 dump. Use ZDDBG/ZDBG command to check the trace entries table to make sure the debug registration entry has the correct workstation IP address. Please note: if the user uses dynamically assigned IP address (DHCP), the IP address for the workstation is changed each time the workstation is connected to the network, so it is very easy to have a debug registration entry that still contains an ‘old’ IP address.

A2: A firewall is preventing the TPF System to communicate to the workstation.

A3: See [how do I set up my workstation IP correctly?](#)

A4: See [why is the debugger not trapping my ECB?](#)

Q5: Why is the debugger not trapping my ECB?

A1: No debug registration entry was registered or registration failed. Use ZDDBG to verify and/or view the TPF Toolkit Console for error messages.

A2: The program to debug is not in the program mask of the debug registration entry. Use ZDDBG to verify.

A3: The workstation IP used to register the debugger was not correct. Use workstation TCP/IP tools to verify that the workstation IP address registered (use ZDDBG to confirm) is correct. Possible ADB01 dump on TPF. Working over VPN, verify the workstation IP address registered is the VPN IP address.

A4: The workstation port is incorrect or the listener failed to start. See debug UI Daemon.

A5: The [conditional registration](#) condition was not met. Attempt to start the debugger without a condition first to verify the program can be debugged at all. Investigate why the condition was not met.

A6: The program was not actually entered to start the debugger. Verify the registered program is actually running (WTOPC, printf, VM trace, etc).

A7: Workstation firewall is blocking the connection, ADB01 dump on TPF.

A8: Ensure registration was performed on the correct TPF system. Use ZDDBG.
Debugging Program

Q1: What is the significance of blue and green font color in my source?

A1: The blue colored source is executable code. The green colored source is comments. If the green colored source is shown where the blue colored source is expected, it is very likely that there is a mismatch between the object and debug information.

Q2: Why is my code not executing sequentially in the debugger?

TPFIC Keywords: ‘debug optimized code’

A1: When the program is built with optimization, the compiler does not generate object code for each source statement sequentially. Since the debugger traces the object code, the user may not see the execution in an orderly manner. See Debugging Optimized code in z/TPF.

Q3: Why are the execute buttons disabled when debugging my program?

A1: The execute buttons are disabled when the highlighted stack frame in the debug view is not the current stack frame. User can click on the current stack frame in the stack area and enable the execute buttons.

A2: The buttons are always disabled if the debugger session is for Dump Viewing or ECB Monitoring because it is viewing static data, not a live ECB.

Q4: Why can’t I step into a function?

A1: The function located inside a module that is marked as NODEBUG in the PAT.

A2: The debugger does not support tracing into the Control Program.

Q5: Why is the debugger saying my program is not in the stepdebug list?

A1: If your program is not in the stepdebug list, it can be added by using the STEPDebug command. After this, if the program can still not be found, make sure the program is in the PAT table by entering ZDPAT <program name> in your TPF System. If no information is shown, the program has not been loaded to your TPF System.

Q6: Why doesn’t the debugger save my source lookup path?

A1: The debugger saves the source lookup path associated with the debug registration entry used to start the debugger in the program profile. If multiple debug registration entries are
used in the RSE, the source lookup path will need to be set up for each. The advantage is that different debug registration entries can be used for specific test systems, test scenarios, or etc.

**A2:** If using a mapped network drive (NFS or SMB), the drive outside of the TPF Toolkit may need to be connected (passwords may need to be entered).

**Q7: Why is TPF Toolkit slow at locating source files?**

**A1:** The source is located on a NFS, SMB, or etc remote drive and the delay is actually in the transfer.

**A2:** The source files are very large. This requires significant processing for the TPF Toolkit to get the line table from the TPF debugger, setting up the executable line map, and etc.

**Q8: Why doesn’t the debugger save my breakpoints?**

**A1:** The debugger saves breakpoints associated with the debug registration entry used to start the debugger in the **program profile**. If using multiple debug registration entries in the **RSE**, then each breakpoint will have to be setup breakpoints for each. The advantage is that different debug registration entries for specific test systems, test scenarios, or etc. can be used.

**A2:** Typically, if the debugger cannot resolve the address of a breakpoint the breakpoint will show as deferred (meaning that the debugger will resolve the breakpoint location when more information about that location is known). However, it may be possible that the breakpoint being set cannot be restored. For example, if a **watch breakpoint** is created on a variable and the debugger is not restarted at a location where the variable can be evaluated.

**A3:** The TPF debugger does not support setting breakpoints from the TPF Toolkit perspective. Any breakpoint set from the TPF Toolkit editor will not show in the TPF debugger.

**Q9: What does the message “Execution may have stopped between statements” mean?**

**Toolkit keywords:** “DBUG9146W” “execution stopped statement” “step return”

**A1:** This statement indicates that the debugger has stopped the application ECB at an assembler instruction that does not perfectly match a line number found in the source view. This is most commonly seen when step-return is used. This message allows the user to step
into the remaining assembler instructions that make up the line. A common example would be debugging C++ code like:

\[
\text{myObj1-&gt;funcToReturnMyObj2-&gt;funcToReturnMyObj3()}
\]

If the user is only interested in debugging the function “funcToReturnMyObj3”, then the user can:
1. Step into the above line.
2. The debugger will stop in function “funcToReturnMyObj2”.
3. Step return.
4. The debugger will stop at the line above just prior to the assembler instructions that call the function “funcToReturnMyObj3”.
5. The debugger displays the above message indicating the application is not stopped at the start of a line.
7. The debugger stops in the function “funcToReturnMyObj3” Using the Debug Console.

**Q10: Why do I get an error when I enter a TPF command on the Debug Console?**

**TPFIC keywords:** “installing LNIATAs for the debug console view” “ZDDBG” “Debug Console view”

**A1:** When a debug console error occurs, a message ID is provided such as “DBUG8001E”. Search the TPF Toolkit help for this id and follow the provided user response.

**A2:** Otherwise a TPF error may have occurred such that the TPF Information Center should be searched for the TPF message ID. A common example is that the LNIATA used does not have CRAS authority to issue the TPF Command. Contact your TPF administrator to have CRAS authority given to the debug console LNIATAs.

**Q11: What does the message "Time Out Occurred: Application may be in an infinite loop. Execution may be between statements." mean?**

**TPFIC keywords:** “MAX_TIME PJ28649”

**Toolkit keywords:** “DBG9057W” “infinite”

**A1:** The debugger time out value can set in the Debug Console View using the TPFTimeout in the Debug Console. See Debug Console View for more information. The amount of time the
application can run before the TPF debugger stops it can be modified by changing MAX_TIME in header file iuddef.h.

**Dump Viewing**

**Q1: Why can’t I see my dump in the dump viewer?**

**A1:** Make sure there is a connection to TPF. An error message “no response from host” appears otherwise.

**A2:** Refresh the dump viewer dump list by right clicking on the dump filter and selecting “refresh” option.

**A3:** Make sure that the filter settings are not filtering out the specific dump. Right clicking on the dump filter and selecting “change” option will bring up the filter settings.

**A4:** Issue a ZDDMP DISP to see if the dump has been captured by TPF.

**Reporting error**

**Q1: What information will IBM require to diagnose the problem?**

**A1:** Each problem is different and requires different information to determine the cause of the problem. Collect the following pieces of information (some may not apply) and send them to IBM:

- Capture a screenshot to show the problem (do not close any views or change anything).
- Give an exacting description of what was done to cause the problem (for example “I did a step over on a function call while having the variables view open”).
- If a dump occurred on TPF collect the dump.
- Collect the portion of the TPF console showing the dump.
- If the dump indicates a module and/object, collect a ZDMAP of module to show the locations of all of the objects in memory.
- [Collect an EPDC trace](#).
- [Collect the .log entry](#).

IBM may ask to run additional traces (like a TPFDB trace), retry with a fix, or etc. Try to explain the problem such that the problem can be recreated (if possible). This is especially
important for problems that IBM cannot recreate as IBM will be relying on the user’s confirmation that the problem is fixed.
Index
1052 state.................................31
ADATA..........................28, 31, 35, 36, 37, 38, 47
ADB01 dump...............................48
ATTAC..............................................15
breakpoints11, 12, 13, 17, 18, 29, 41, 42, 43, 50
breakpoints..........................13, 14, 23
entry breakpoints.........................12, 18, 42
line breakpoint............................13, 20
load breakpoint............................13
macro breakpoint.........................14, 41
macro group breakpoint...................14, 41
watch breakpoint..........................13
build options............................29
CTEST.................................18, 27, 34
cetest()..............................................27
deploy command............................15, 45
dump command.............................15, 45
LNIATA................................16, 51
TPF command................................15, 45, 51
dump information.27, 28, 29, 31, 32, 35, 36, 38, 39, 40, 41, 47, 49
dump registration..........................8, 33, 47
DETAC.................................................15
dump capturing.............................10
manage dumps on TPF.....................45
dump debugging.............................
dump viewer.................................10, 45, 52
event breakpoints..........................12
general features and techniques........41
dump viewer.................................8, 10, 11, 31, 42, 45, 52
ecb monitor.................................8, 9, 45
ecb monitor.................................
access ECB list.........................8, 9, 45
add filters.................................9
obtain ECB snapshot.......................9
set properties.............................9
ECBHeap........................................16
ECBSnapshot.................................10, 16
ECBTrace........................................16
eclipse basics..............................7
EPDC Trace.......................................46
FLIPC..............................................15
Follow child...................................18
Follow parent..................................18
Fork...............................................18
GETCC.............................................15
global expressions..........................41
Global Symbols.............................30
heap check...............................17, 31
load breakpoint............................13
macro.............................................42
macro.............................................
ASM..............................................41
C/C++..............................................27
ecbtrace.........................................16
trlog.............................................16
UCST.............................................32
memory breakpoint.......................14, 42
memory mapping (rendering)..............20, 22
optimized code............................15
record hold....................................14
register the debugger........................8, 33, 47
RELCC.............................................15
Report debug error..........................46, 52
source lookup path..........................47
stack frame...............................17, 21, 49
STEPDebug.................................16, 17, 35, 49
system error...................................12
TPF debugger trace..........................46
TPF IP address.................................34
TPF Toolkit .log...............................46
TPFDGTrace.................................16
TPFSYM.......................................31, 32, 37, 38
TPFTimeout.................................16, 32
TPFXMLGEN.................................25, 31, 40
TRLOG.............................................16, 42
UCST...............................................24, 28, 31, 32
workstation IP address....................17, 35, 48
XCPTRAP...........................................13, 16
XML files.................................14, 24, 25, 31, 40
XML files.................................
memory mapping (rendering)..............22
XML files - TPF mappings

- eb0eb.xml ........................................ 15, 40
- sw00sr.xml ........................................ 25
- DataLevelDisplay.xml ...................... 15
- DECBDisplay.xml .............................. 19
- ECB.xml ........................................ 19
- idecb.xml ........................................ 19
- sw00sr.xml ........................................ 25
- SW00SRSummaryTable.xml ................. 25
- ZAPAT ............................................ 27, 39
- ZDBG ............................................. 8, 48
- ZDDDBG ........................................... 8, 39, 48, 51
- ZDDMP ........................................... 10, 11, 45, 52
- ZDDMP ........................................... 10, 11, 45, 52
- ZDAP .............................................. 19
- ZDFIL ............................................ 37, 38
- ZDMAP ............................................ 38, 39, 52
- ZDPAT ............................................ 36, 37, 49
- ZDPGM ............................................ 36
- ZDREC ............................................ 37, 38
- ZINET ............................................ 8, 10
- ZSTRC ............................................ 31
- #define ......................................... 27, 40, 41
Notices

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A  

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Department 830A  
Mail Drop P131  
2455 South Road  
Poughkeepsie, NY 12601-5400  
U.S.A  

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM


Other company, product, or service names may be trademarks or service marks of others.