IBM 4765 PCIe Cryptographic Coprocessor
ICAT Debugger
Getting Started
Note: Before using this information and the products it supports, be sure to read the general information under "Notices" on page 17.

Fifth Edition (June, 2014)
This and other publications related to the IBM 4765 PCIe Cryptographic Coprocessor can be obtained in PDF format from the product Web site. Click on the PCIe Cryptographic Coprocessor link at http://www.ibm.com/security/cryptocards, and then click on the Library link.

Reader's comments can be communicated to IBM by contacting the Crypto team at crypto@us.ibm.com.

© Copyright International Business Machines Corporation 2011, 2014.
Note to U. S. Government Users—Documentation related to restricted rights Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Figures

Figure 1 Launch or attach window - Launch page ................................................................. 9
Figure 2 Launch or attach window - Attach page ............................................................... 10
Figure 3 Debug Session Control window ........................................................................ 15
About this manual
This document contains information to help you install and get started with the Interactive Code Analysis Tool (ICAT) debugger supplied with the IBM 4765 Developer’s Toolkit. This document is intended for use as a quick reference. For more specific details, refer to the ICAT Debugger Help available from the Help menu of the debugger.

If you need assistance from any window while using the debugger, press F1 while viewing a window or choose an item from the Help menu.

This manual should be used in conjunction with the manuals listed under “Related publications” in this section.

Prerequisite knowledge
The reader of this book should understand how to perform basic tasks (including editing, system configuration, file system navigation, and creating application programs) on the host machine and in the Linux® environment. Familiarity with the IBM 4765 application development process (as described in the IBM 4765 PCIe Cryptographic Coprocessor Custom Software Developer’s Toolkit Guide) is also required.

Organization of this manual
This book is organized as follows:
“Introducing the ICAT debugger” discusses a sample debug session for a coprocessor-side application.
“Main window” describes the components of the Debug Session Control interface.
“Supported expressions” introduces the expression language supported by ICAT.
“Notices” includes product and publication notices.
An index completes the manual.

Typographic conventions
This publication uses the following typographic conventions:
- Commands that you enter verbatim onto the command line are presented in monospace type.
- Variable information and parameters, such as file names, are presented in italic type.
- Constants are presented in bold type.
- The names of items that are displayed in graphical user interface (GUI) applications, such as pull-down menus, check boxes, radio buttons, and fields, are presented in bold type.
- Items displayed within pull-down menus are presented in bold italic type.
- Function names are presented in italic type.
- System responses in a shell-based environment are presented in monospace type.
- Web addresses and directory paths are presented in italic type.

Syntax diagrams
The syntax diagrams in this section follow the typographic conventions listed in “Typographic conventions.” Optional items appear in brackets. Lists from which a selection must be made appear in braces with vertical bars separating the choices. See the following example.

COMMAND  \texttt{firstarg [secondarg] \{a \mid b\}
A value for firstarg must be specified. secondarg may be omitted. Either a or b must be specified.

**Related publications**


**Summary of changes**

This edition of IBM 4765 PCIe Cryptographic Coprocessor ICAT Debugger Getting Started contains product information that is current with the IBM 4765 PCIe Cryptographic Coprocessor announcements.
Introducing the ICAT debugger

ICAT is a source-level debugger that enables developers to debug applications running on an IBM 4765 Cryptographic Coprocessor (referred to in this document as the IBM 4765). The debugger provides a graphical user interface that enables a developer to:

- locate the current point of execution within an application and view the source that corresponds to that location,
- examine and modify an application’s state, including variables and registers,
- set breakpoints and execute machine instructions or source statements one-by-one,
- dump the contents of the call stack, and
- intercept and diagnose exceptions generated by an application.

The ICAT debugger (hereafter referred to in this document as the debugger) is a source-level debugger that runs on a machine from the IBM ServerProven™ list running a Linux operating system that is supported for the current release of the toolkit. See the System x section of the Software download page on the IBM PCIe Cryptographic Coprocessor Web site for details:


Before you begin

This section lists the hardware and software requirements, options that can be used when compiling and linking your program, environment variables, and the search order of source files and modules.

Minimum hardware requirements

- A server from the IBM Server-proven list, available here:

- If the debugger is to communicate with the IBM 4765 adapter using TCP, a special shaved cable and a dongle are required. See the IBM 4765 Cryptographic Coprocessor Custom Software Developer’s Toolkit Guide for details.

Installation


Environment variables

The debugger uses environment variables to manage debugging sessions and remote communication. Use one of the following methods to set the environment variables:

- Create and modify your own command file with the environment variables you want to set. See "Helpful tips and hints" on page 11 for more information.
  or
- Set the environment variables in the session from which the debugger is started by selecting the Settings button from the Launch or attach window.
Refer to the ICAT Debugger Help (Getting Started->Before You Begin->Environment Variables) for a list of the available environment variables and a description of each.

Typically users will set the following environment variables:

**CAT_COMMUNICATION_TYPE**
Specifies how the host portion of the debugger should communicate with the coprocessor-side debugger daemon. Both PCI and TCP modes of communication are supported. Most customers will prefer to use PCI communication, as it requires less setup. TCP communication is primarily used when the debugger resides on a different machine from the adapter on which the application being debugged is loaded.

Valid values: TCP, PCI

**CAT_MACHINE**
Specifies the location of the 4765 containing the application to be debugged.

When using PCI communication:

\[0, N]\) where \(N\) is the number of 4765 adapters present in the host system - 1. Please note that adapter numbers are zero-based. Therefore the first adapter is adapter 0.

When using TCP communication:

IP Address:Port  (Example 10.1.0.1:3535)

*Note:* When you specify a port number (\(N\)) for use in TCP mode, ports \(N\) and \(N+1\) are used. ICAT requires port \(N+1\) for a halt thread that allows the debugger to stop execution while the program is running. Therefore, if you need to open a port for ICAT in your firewall, you must open both ports \(N\) and \(N+1\).

**CAT_HOST_BIN_PATH**
Specifies where ICAT can find a local copy of the application being debugged.

**CAT_HOST_SOURCE_PATH**
Specifies where ICAT should look for source files for the executable being debugged.

**CAT_PATH_RECURSE**
Specifies the top-most directory in a directory tree where ICAT should look for source files for the executable being debugged.

**Finding source files**
The debugger searches for the source files in the following order:

1. CAT_OVERRIDE environment variable, if specified.
2. The subdirectory in which the object file generated from the source was compiled (as indicated by debug information in the executable file).
3. Host binary path or CAT_HOST_BIN_PATH environment variable, descending subdirectories if the CAT_PATH_RECURSE environment variable is set or Recursive file searching is selected on the Remote page of the Debugger Properties window.
4. Host source path or CAT_HOST_SOURCE_PATH environment variable, descending subdirectories if the CAT_PATH_RECURSE environment variable is set or Recursive file searching is selected on the Remote page of the Debugger Properties window.
5. The current subdirectory.
6. The path defined in the INCLUDE environment variable.

7. The last specified subdirectory from the Change Text File window.

8. If the debugger cannot find the source in any of the previously mentioned locations, it prompts the user to enter the location of the required source file.

The debugger searches for executable files in the following order:


2. Host binary path or CAT_HOST_BIN_PATH environment variable, descending subdirectories if the CAT_PATH_RECURSE environment variable is set or Recursive file searching is selected on the Remote page of the Debugger Properties window.

Limitations

The debugger has the following restrictions:

- The debugger can either launch an application (that is, cause an application to be loaded into the cryptographic coprocessor and assume control of the application before any instructions in the application have been executed) or attach to an application. The earliest point at which the debugger can attach to an application (that is, assume control of the application and place it under debug) is after the application’s main entry point has been started. If you want to make certain that your application does not make progress before the debugger has a chance to attach, you must code an infinite loop at the beginning of the application (and use the debugger to change the point of execution to the statement following the loop after attaching).

- Applications to be debugged must be compiled and linked in such a way that the application executable incorporates debug information. Otherwise, the debugger cannot be used to examine and manipulate the application at the source level. However, only the copy of the application executable that the debugger reads must have this information; the copy downloaded to the coprocessor can be reduced in size by stripping debug information from it before it is downloaded.

- To source-level debug your application, you must compile and link your application with debugging data. You must use \texttt{–gstabs+} when you compile.

The debugger does not handle certain coding styles well. For example, it can be difficult to debug a program that has more than one source statement on a line — the debug information available to the debugger forces the debugger to treat the entire line as a single statement. Thus, you cannot set a breakpoint on, for example, the second statement on the line nor can you step through each statement.

This is not technically a limitation of the debugger, but rather in of how debug information is made available to it. Debug information is presented on a line-by-line basis, and if there are multiple statements on a single line, the debugger must jump over all of them when the step (line) function is called.

- The 4765 Linux device driver has a default timeout of 30 seconds, and it issues a reset to the adapter when it detects a request has not received a reply within the timeout interval. The default timeout may not provide sufficient time for the developer to debug the coprocessor-side piece of a coprocessor application since the application will be delayed in sending the reply while a developer is debugging.

The Toolkit includes scripts (\texttt{y4unload/y4load}) that are used to load and unload the device driver. These scripts must be run as root or via sudo.

\texttt{y4load} loads the device driver:
sudo y4load

A developer may want to add y4tk/<version>/bin/host/ to the PATH so that these scripts are easily accessible.

The process for changing the default host device driver timeout for the 4765 has changed as compared to the process for the 4764. On SLES11, the file /etc/modprobe.conf.local must be amended to specify the timeout options passed to the 4765 host device driver. Options passed to the y4load script are overridden by the settings in /etc/modprobe.conf.local on SLES-based systems. This means that specifying a timeout value as a parameter to the y4load script will have no effect on the timeout value.

Note: To see the current value of the ycrypto device driver timeout values, issue the following command:

cat /proc/driver/ycrypto/timeout

Before making any changes to /etc/modprobe.conf.local, MAKE A BACKUP COPY OF THIS FILE AND STORE IT IN A SAFE LOCATION!

Then, as root, edit /etc/modprobe.conf.local to indicate that the install ycrypto line specifies the desired timeout values:

install ycrypto /sbin/modprobe --ignore-install ycrypto
opcodes_dir=/opt/IBM/4765 timeout=<Request Window 1 Timeout Value>,<Request Window 2 Timeout Value>,<Request Window 3 Timeout Value>,<Request Window 4 Timeout Value> ssp_timeout=<SSP timeout value> mb_timeout=<Miniboot timeout value>; /sbin/ycrypto_mknod

where all timeout values are specified in seconds. For example, if you wanted to have the timeout value set to 1200 seconds, specify the following line in /etc/modprobe.conf.local:

install ycrypto /sbin/modprobe --ignore-install ycrypto
opcodes_dir=/opt/IBM/4765 timeout=1200,1200,1200,1200
ssp_timeout=1200 mb_timeout=1200 ; /sbin/ycrypto_mknod

Once this file has been modified to reflect the desired timeout, you need to unload and reload the device driver (as root). This can be done with the y4unload and y4load scripts in the toolkit, or you can use the modprobe command to unload and load the driver:

modprobe -r ycrypto

modprobe ycrypto

To verify the timeout value has been changed, issue this command:

cat /proc/driver/ycrypto/timeout

---

**Getting started**

This section describes how to set up the coprocessor and the host.

**Setting up the coprocessor**

To set up the coprocessor:

1. Follow the instructions for installation in the *IBM 4765 Cryptographic Coprocessor Custom* 4

4 IBM 4765 ICAT Debugger Getting Started
Software Developer’s Toolkit Guide to install the coprocessor and to prepare the coprocessor for use as a development environment.

2. Ensure that the segment 3 initialization shell script includes the following:

- If running a Toolkit (non-UDX) application, the Outbound Authentication daemon, `xcoad`, must be started before `ydaemon` in order for `ydaemon` to run. A typical invocation would be:
  ```bash
  /ramS3/0/xcoad
  ```

- If running a UDX, `startcdud` must be started. This launches the OA daemon for UDX-based applications. A typical invocation would be:
  ```bash
  /ramS3/0/startcdud
  ```

3. Follow the instructions for the IBM 4765 listed below, depending on the method you want the debugger to communicate with the IBM 4765. (A communication type must be specified.)

The changes should be incorporated into the init.sh segment 3 initialization shell script (provided with the Toolkit) which is incorporated into a JFFS2 image loaded onto the adapter using DRUID.

Note: The Toolkit contains ready-to-use scripts for PCI communication, and requires no changes when debugging the test UDX or sample Toolkit applications. See the `y4tk/<version>/shells` directory for example shells.

- **PCI**
  a. Set the following environment variable to specify PCI communication:
     ```bash
     export CAT_COMMUNICATION_TYPE=PCI
     ```
  b. Start the debugger daemon by issuing:
     ```bash
     <fully qualified path>/ydaemon &
     ```
     Note: Typically, this is `/ramS3/0/ydaemon`.

- **Ethernet**
  a. Issue the following commands before launching the coprocessor-side debugger daemon:
     ```bash
     ifconfig eth0 <IP address> netmask <appropriate netmask> route add default gw <default gateway IP address>
     ```
  b. Set the following environment variable to specify communication by way of an Ethernet card:
     ```bash
     export CAT_COMMUNICATION_TYPE=TCP
     ```
  c. Start the debugger daemon by issuing:
     ```bash
     <fully qualified path>/ydaemon nnnn &
     ```
     where `nnnn` is a port number, of your choosing, greater than 1024.
     Note: Typically, the fully qualified path is `/ramS3/0/ydaemon` and the default port number is 3535.

4. Use the DRUID utility to load your application and start it running for debugging. The application should have an infinite loop near the beginning of the code, as recommended in the *IBM 4765 Cryptographic Coprocessor Custom Software Developer’s Toolkit Guide*.

**Warning**

ICAT requires the Outbound Authentication daemon to be running on the adapter before `ydaemon` is
Setting up the host computer
To set up the host Linux computer:

1. Install the IBM 4765 Toolkit. The debugger is packaged with the Toolkit, and is located in the
   y4tk/<version>/debuggers/icatpyx<arch>-<version> directory of the installation image. Please
   note that there may be two versions of ICAT in the toolkit. For convenience, IBM provides
   versions of ICAT that run as either a 32 or 64 bit application. The card-side app, of course, is
   always a 32-bit application. Two versions are provided because some 64-bit operating systems
   do not provide the necessary 32-bit support required to run ICAT as part of a default install. Both
   the 32 and 64 bit versions of ICAT provide equivalent functionality.

2. Ensure that <path to debugger>/icatpyx<arch>-<version>/bin is in the path.

3. Set the environment variables. See “Environment variables” on page 1 and “Helpful tips and
   hints” on page 11 for more information.

4. Follow the instructions listed below to specify the communication type:
   - PCI
     a. Set the following environment variables to specify PCI communication:
        ```
        export CAT_MACHINE=<N>
        where N is the adapter; the first adapter in the system is numbered 0.
        export CAT_COMMUNICATION_TYPE=PCI
        ```
   - Ethernet
     a. Set the following environment variables to specify communication by way of an Ethernet
        card:
        ```
        export CAT_MACHINE=<IP address>:<port number>
        where IP address and port number must match what was specified in the segment 3
        initialization script.
        ```
        **Note:** When you specify a port number (N) for use in TCP mode, ports N and N+1 are
        used. ICAT requires port N+1 for a halt thread that allows the debugger to stop execution
        while the program is running. Therefore, if you need to open a port for ICAT in your
        firewall, you must open both ports N and N+1.
        ```
        export CAT_COMMUNICATION_TYPE=TCP
        ```

Demonstration session
The following session demonstrates the debugger manipulating the code on the coprocessor.

1. Locate the y4tk/<version>/samples/toolkit/rte subdirectory on your host computer. This
   subdirectory contains the C source files and the makefiles to make the binaries that the debugger
   must see on your host computer.

2. Export the following environment variables:
   - Y4TK_FS_ROOT must be set to point to the root of the Developer's Toolkit (that is, the fully
     qualified path to y4tk/<version>). For example, if the Toolkits are unzipped/untarred from
     /home/user, then Y4TK_FS_ROOT becomes /home/user/y4tk/<version>.
CROSS must be set to the root directory containing the cross-targeted compiler, assembler, and loader. For example, if the cross-compiler is located in /opt/cross/ppcnf, CROSS becomes /opt/cross/ppcnf, as in export CROSS=/opt/cross/ppcnf.

GCC_NAME must be set to the prefix to use with the standard compiler, assembler, and loader names to create the corresponding cross-targeted tool names. For example, GCC_NAME is ppcnf-linux-, as in export GCC_NAME=ppcnf-linux-.

ICAT_FS_ROOT must be set to point to the root of the ICAT installation (that is, the fully qualified path to y4tk/<version>/debuggers/<platform>/icatpyx<arch>-nnn, where arch is the architecture of the host machine (32 or 64 bit) and nnn is the version of the debugger and where platform specifies the OS being used). For example, if the Toolkits are unzipped/untarred from /home/user on Linux, then ICAT_FS_ROOT becomes /home/user/y4tk/<version>/debuggers/linux/icatpyx<arch>-nnn.

Y4_JFFS2_DIR must be set to point to the directory that contains the mkfs.jffs2 utility. For example, if mkfs.jffs2 is installed in /usr/sbin, then Y4_JFFS2_DIR becomes /usr/sbin.

3. Make the coprocessor-side executable using the makefile in y4tk/<version>/samples/toolkit/rte/card/gcc. A typical invocation would be:

```
make -f card.mak DEBUG=y
```

The executable is saved at y4tk/<version>/samples/toolkit/rte/card/gcc/sampleCardApp.

4. Use the y4tk/<version>/samples/toolkit/rte/host/gcc/host.mak file to make the sampleHostApp executable for the host computer. A typical invocation would be:

```
make -f host.mak
```

The host executable is saved in y4tk/<version>/samples/toolkit/rte/host/gcc/sampleHostApp.

5. Build the segment-3 JFFS2 image. If PCI communication is used, no changes are required to the segment-3 initialization shell script. If TCP communication is used, the segment-3 initialization shell script must meet all requirements for TCP communication listed in "Setting up the coprocessor" on page 4.

```
cd $Y4TK_FS_ROOT/build_seg3_image
make -f y4tk.seg3.image.mak SAMPLE_NAME=rte BUILD_TYPE=debug
```

The segment 3 JFFS2 image will be located in y4tk/<version>/build_seg3_image/y4tk.rte.<DATE>.bin.

Note: when you build the image, the makefile also creates a shell script you can use to set the ICAT environment variables for the sample. This script is placed in the $Y4TK_FS_ROOT/build_seg3_image directory. The script assumes PCI communications and that the sample will be loaded into adapter 0 (the first adapter). See the y4tk.seg3.image.mak file for more information.

6. Use DRUID to load the mkfs image onto the coprocessor. Invoke DRUID as:

```
druid <segment 3 JFFS2 image name> [adapterNumber]
```

7. On the host computer, export the CAT_HOST_SOURCE_PATH, CAT_HOST_BIN_PATH, CAT_COMMUNICATION_TYPE and CAT_MACHINE environment variables to reflect your environment. Run icatpyx, and then wait for the Launch or attach window to display.

```
export CAT_HOST_SOURCE_PATH=y4tk/<version>/samples/toolkit/rte/card/
export CAT_HOST_BIN_PATH=y4tk/<version>/samples/toolkit/rte/card/gcc/
export CAT_COMMUNICATION_TYPE=PCI
export CAT_MACHINE=<N>
```

Introducing the ICAT debugger 7
where <version> is the toolkit version number and <N> is the adapter number (zero-based).

Or, run the setup script that was created when the image was built. Then run icatpyx, and then wait for the Launch or attach window to display.

. seticat_rte.sh
icatpyx

8. Enter sampleCardApp in the Program field on the Attach page of the Launch or attach window. Click OK to attach the program. Wait for the Debug Session Control window to be displayed with the sampleCardApp application in the component list.

   Note: When attaching, specify ONLY the process name. Do not enter path information.

9. Click the twistie beside the path name for the executable file in the Debug Session Control window. The path expands to display a list of source files in the executable file. Click the twistie beside rteX. A list of functions is displayed. Double-click main to display the function in the Source window.

10. Set a breakpoint in the debug spin loop and then click Run.

11. Jump to the call to xcAttachWithCDUOption and then step over the call. At this point, you can debug the program normally. If you click Run before stepping over xcAttachWithCDUOption, the system stops inside xcGetRequest because no host function has asked for service yet. Since xcGetRequest has no source level debugging information, ICAT will show a disassembly window if it is stopped inside xcGetRequest. To determine exactly where the execution point is in the call stack, use the Call stack window in the Monitors menu.

12. Run the sampleHostApp application from the host.

   Note: You must have stepped past the call for xcAttachWithCDUOption for sampleHostApp to run properly.

13. Display a Mixed view when the debugger hits a breakpoint. Single step the assembler code a couple of times. Switch back to a Source view. Next, set a breakpoint at the check for the return code of xcGetRequest (if( rc < 0 ) ), and run again. When you hit that breakpoint, you can double-click variables, do a call stack unwind, show a Register window or a Storage window, and so on.

This concludes the demonstration session.

Starting a debug session
Load the program you want to debug on the coprocessor. To start the debugger:

From the Linux command prompt, enter icatpyx.

The Launch or attach window is displayed. Typically, if you started your card-side application in init.sh, choose Attach. If you did not start your card-side application in init.sh, you would typically choose Launch.

   Note: you must always consider the state of your adapter when trying to launch or attach.

So, if your are trying to attach after already running, launching, or attaching to your application, you must take into account the current state of your application. This state affects your decision to launch or relaunch, attach, CLU RS, or even reload segment 2 and segment 3 of the adapter.

   Note: It's usually convenient to have set any environment variables you want to specify through a shell script (for example, seticat.sh) before starting the debugger. See “Environment variables” on page 1 and

8 IBM 4765 ICAT Debugger Getting Started
Launching your program

Use Launch for debugging a program that has not been started or to restart a program that has been terminated successfully. If launching, you will typically not include your card-side program in init.sh.

1. Select the Launch tab. The Launch page is displayed as shown in Figure 1. This is the default page. If launching, do not start sampleCardApp in init.sh.

   ![Launch or attach Window](image)

   Figure 1 Launch or attach window - Launch page

2. Enter the name of the program to start in the Program field.
   Note: When launching, the program name must include the full path to the application.

3. Enter any parameters you want to start the program in the Parameters field.

4. Select Use program profile to reactivate the windows and breakpoints, if you’re going to debug a program more than once.

5. Select Debug program initialization if you want to debug the initialization code for the program.

6. Click Settings to display the Debugger Properties window if you want to set how threads and source files are initially displayed. Click the Source tab to view the Source page and select the changes that you want. If you make any changes, click Apply. Click Close to close the window.

7. Click OK to start debugging the program. The Debug Session Control window is displayed showing the threads and components of your program.
Reset returns the window settings to the values you defined upon initialization of the window. The Default button restores the window’s default settings.

Settings displays the Debugger Properties window, which enables you to select how threads and source files are initially displayed and enables you to set environment variables. Refer to “Setting Debugger Properties” located within the ICAT Debugger Help for more information.

Attaching to a program that is running
Use Attach to attach to a card-side program that is running on the adapter.

1. Select the Attach tab. The Attach page is displayed as shown in Figure 2.

![Launch or attach window - Attach page](image)

2. Enter the name of the program to start in the Program field.
   Note: This name is the process name only. Do not include path information.

3. Select Use program profile to reactivate the windows and breakpoints if you’re going to debug a program more than once.

4. Click OK to attach to the program.

Note: If you debug your program more than once, it will be in a different place when you attach to it after the first time. To see a call stack unwind that will allow you to locate the execution point in relation to “your” code, use the Call stack window in the Monitors menu. Use this information to set appropriate breakpoints in your program.
Using the tool buttons

A tool bar has been provided on the debugger windows for easier access to frequently used features. To display buttons in a window, enable the Tool buttons choice that is listed under the Options menu. A list of the available tool buttons and their features is included in the ICAT Debugger Help (Getting Started -> Using the Tool Buttons).

Helpful tips and hints

The following tips and hints may be helpful:

- You must have the correct Linux installation and version on your host computer.
- Put any environment variables that you want set in a command file which you would start before running the debugger. For example:

  1. Create a script (for example, seticat.sh) which contains the environment variables you want set. Typical settings would be:
     
     For PCI:
     
     ```
     export ICAT_BROWSER=<path to browser>
     export CAT_COMMUNICATION_TYPE=PCI
     export CAT_MACHINE=0
     export CAT_HOST_BIN_PATH=    
     y4tk/<version>/samples/toolkit/rte/card/gcc
     export CAT_HOST_SOURCE_PATH=    
     y4tk/<version>/samples/toolkit/rte/card
     ```

     For TCP:
     
     ```
     export ICAT_BROWSER=<path to browser>
     export CAT_COMMUNICATION_TYPE=TCP
     export CAT_MACHINE= <IP:Port>
     export CAT_HOST_BIN_PATH=    
     y4tk/<version>/samples/toolkit/rte/card/gcc
     export CAT_HOST_SOURCE_PATH=    
     y4tk/<version>/samples/toolkit/rte/card
     ```

  2. Ensure the command file is executable by issuing:

     ```
     chmod +x seticat.sh
     ```

  3. Start the command file by using a period and a space before the filename:

     ```
     . seticat.sh
     ```

  4. Run the debugger by issuing:

     ```
     icatpyx
     ```

- Using C, you can write your program code with stylistic features that are not supported by the debugger. For example, multiple statements on the same line are difficult to debug. None of the individual statements can be accessed separately when you set breakpoints or when you use step commands.

- When stopping execution of a running program, the execution will often be halted in a location outside of the debuggable source of your toolkit application or UDX. To see a call stack unwind that will allow you to locate the execution point in relation to "your" code, use the use the Call stack window in the Monitors menu.

- The CLU RS command can be used to reset the adapter and rerun the init.sh file. This is useful
when you want to debug an application a second time and need to stop in the debug spin loop. If the application has already been run once under the debugger, its execution point is most likely already past the debug spin loop. Issue the CLU RS command to get the application back to the state where you can stop in the debug spin loop.

**Troubleshooting**

Use the following checklist if you're having problems starting a remote debug session.

1. Make certain that the debugger daemon and application have been loaded onto and started on the coprocessor. See “Setting up the coprocessor” on page 4 for details.

   Note: When `ydaemon` starts, it will log a message to the `/var/log/messages` file on the host. The command `sudo tail -f /var/log/messages` will display the contents of this file. You can use `sudo tail -f /var/log/messages | grep -i ydaemon` to see only the `ydaemon` messages.

2. Wait several minutes if you have just loaded your code onto the coprocessor. The adapter may take some time to run the program to the point at which it can be attached.

3. If using TCP communication, verify with your network administrator that all network settings, on both your host computer and on the 4765, are correct. Also ensure there are no firewalls preventing communication. When you specify a port number (N) for use in TCP mode, ports N and N+1 are used. ICAT requires port N+1 for a halt thread that allows the debugger to stop execution while the program is running. Therefore, if you need to open a port for ICAT in your firewall, you must open both ports N and N+1.

4. Ensure the following, if you want to launch an application, and specify the application name and arguments on the command line:
   a. The value of the `CAT_COMMUNICATION_TYPE` environment variable is set to the communication mode you are using. This value must be set on both the host and the coprocessor.

      | Value | Description |
      |-------|-------------|
      | TCP   | Ethernet communication |
      | PCI   | PCIe bus communication |

   b. The value of the `CAT_MACHINE` environment variable is set to the appropriate value for the communication mode you are using. This value only needs to be set on the host.

      | Communication mode | Value |
      |-------------------|-------|
      | TCP               | The IP address of the adapter, followed by a colon, followed by the port number used by the daemon on the coprocessor. |
      | PCI               | The PCI adapter number |

   c. You have a readable copy of the relevant executable files for your application on the host computer and the directory paths to the files are listed in the value of the `CAT_HOST_BIN_PATH` environment variable.

   d. You have a readable copy of the relevant source files for your application on the host computer and the directory paths to the files are listed in the value of the `CAT_HOST_SOURCE_PATH` environment variable.
e. You set the **CAT_PATH_RECURSE** environment variable, if needed, to find the copy of the executable or source files for your application on the host computer.

f. You specify the application name and arguments correctly when launching or attaching.

5. Use the Launch or attach window, if you do not specify the application name and arguments on the command line, as follows:

a. Click **Settings**. The Debugger Properties window is displayed.

   1) Click the Remote tab to view the Remote page.

   2) Ensure that the Communication mode field is set according to the type of communication you are using.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Ethernet communication</td>
</tr>
<tr>
<td>PCI</td>
<td>PCIe bus communication</td>
</tr>
</tbody>
</table>

   **Ethernet communication:**
   - Ensure that the **IP address** entry field is set to the IP address of the adapter.
   - Ensure that the **Port #** entry field is set to the port number used by the daemon on the coprocessor.

   **PCI communication:**
   - Ensure that the **PCI** field is set to adapter number of the IBM 4765.

   3) Make certain that you have a readable copy of the relevant executable files for your application on the host computer and the directory paths to the files are listed in the **Host binary path** entry field.

   **Note:** The value in this field comes from the **CAT_HOST_BIN_PATH** environment variable.

   4) Ensure that you have a readable copy of the relevant source files for your application on the host computer and the directory paths to the files are listed in the **Host source path** entry field.

   **Note:** The value in this field comes from the **CAT_HOST_SOURCE_PATH** environment variable.

   5) Select **Recursive file searching**, if needed, to find the copy of the executable or source files for your application on the host computer.

   **Note:** The **CAT_PATH_RECURSE** environment variable is used for recursive file searching.

   6) Make certain that the **Remote binary path** is set if you need the daemon to launch the application from a directory other than the current directory for the session running the daemon.

   7) Click **Apply** if you made any changes on the window.

   8) Click **Close** to return to the Launch or Attach window.
b. Ensure that you specify the correct program name in the **Program** entry field.

c. Ensure, if launching your application, that you specify any arguments to your application in the **Parameters** entry field located on the **Launch** page.

d. Make certain, if attaching to your application, that the application is currently executing on the target computer.

6. If you attempt to attach to an application and receive an error message, make sure you have followed the setup procedures in “Setting up the coprocessor” on page 4 and in “Setting up the host computer” on page 6 and have correctly compiled, linked, and loaded the coprocessor-side application. Also ensure that the *init.sh* copied from the *y4tk/<version>/shells* directory has not been modified.

**Ending the debugging session**

To end the debugging session, click **Close debugger** (located within the **File** menu) from any of the debugger windows. The **Close Debugger** window is displayed. Select one of the following choices:

- Click **Yes** to end your debugging session.
- Click **No** to return to the current debugger window without exiting the debugger.

You can also end the debugging session by pressing F3 in any of the debugger windows.
Main window

The Debug Session Control window is the control window of the debugger and is displayed during the entire debugging session. This window is divided into two panes: Threads and Components. See Figure 3 for an example.

The Threads pane contains the threads, their names, and the state of the threads started by your program. To display the state of a thread, click the plus icon located to the left of the thread.

Right-click a selected item to display the Thread menu and press F1 to view help for this item.

The Components pane shows the path names of the modules that you are debugging. Right-click a selected item to display the Component menu and press F1 to view help for this item.

Refer to the Windows section of the ICAT Debugger Help for descriptions and use of the Debug Session Control window menus, and other windows available from the Debug Session Control window and a description of their menus.

Managing fonts

All ICAT windows except the Source window use the Adobe Courier 12 font. To change the default font used in your sessions (including ICAT), modify the .gtrc-2.0 file in your home directory (or create the file if it does not exist), and add the following line:

```
gtk-font-name = "<font string>"
```

Some examples include:

```
gtk-font-name = "Adobe Courier 14"
gtk-font-name = "Times Roman 12"
gtk-font-name = "Schumacher Clean 12"
```
Supported expressions

The expression language supported by the debugger, which is a subset of C, includes the operands, operators, and data types.

**Note:** You can display and update bit fields for C code only. You cannot look at variables that have been defined using the #DEFINE preprocessor directive.

Refer to the **Expressions Supported** section of the ICAT Debugger Help for details about the expressions supported.
Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594, USA.

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.

Copying and distributing softcopy files

For online versions of this book, we authorize you to:
Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.
Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.
You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both: AIX IBM

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

- attaching to a program ........................................ 10
- changing the host device driver timeout ..................... 4
- coprocessor, setting up ........................................ 4
- debug session control window .................................. 15
- demonstration session ........................................... 6
- ending the debugging session ................................... 14
- environment variables .......................................... 1
- expressions .......................................................... 16
- finding source files ............................................... 2
- getting started ..................................................... 4
- hardware requirements ........................................... 1
- helpful tips and hints ............................................. 11
- host computer, setting up ........................................ 6
- ICAT Debugger ..................................................... 1
- installation .......................................................... 1
- interactive code analysis tool (ICAT) ......................... 1
- Introduction ......................................................... 1
- launching your program ......................................... 9
- limitations .......................................................... 3
- main window ....................................................... 15
- managing fonts ..................................................... 15
- prerequisite knowledge ......................................... vi
- related publications ............................................. vi
- restrictions ........................................................... 3
- setting up the coprocessor ...................................... 4
- setting up the host computer ................................... 6
- starting a debug session ........................................ 8
- tool buttons ....................................................... 11
- troubleshooting .................................................. 12
- Notices ............................................................. 17