IBM Visual Performance Analyzer

User Guide

Version 5.0

Issue Date: 06/06/2007

Revision Status: Final

About this Document

This document describes how to install and use the Visual Performance Analyzer tool. This document will help you install the tool, learn how to collect performance data on your platform and later analyze the data, using the VPA plug-ins.

Table of Contents

1.Introduction

What is Visual Performance Analyzer?

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. It consists of five major components: Profile Analyzer, Code Analyzer, Pipeline Analyzer, Counter Analyzer and Trace Analyzer.

Profile Analyzer provides a powerful set of graphical and text-based views that allow users to narrow down performance problems to a particular process, thread, module, symbol, offset, instruction, or source line. Profile Analyzer supports time-based system profiles (Tprofs) collected from a number of IBM platforms.

Code Analyzer examines executable files and displays detailed information about functions, basic blocks, and assembly instructions. It is built on top of FDPR-Pro (Feedback Directed Program Restructuring) technology and allows adding of FDPR-Pro and Tprof profile information. (The Linux® version of FDPR-Pro is available here at AlphaWorks.) Code Analyzer is able to show statistics; navigate disassembled instructions; and display performance comments, instruction grouping information, and map instructions back to source code.

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another AlphaWorks technology. Pipeline joins the VPA toolkit to provide VPA users with the means of examining how code is executed on various IBM POWER processors. Pipeline Analyzer displays the pipeline execution of instruction traces generated by a POWER series processor. It does so by providing a scroll view and a resource view of the instruction execution.

Counter Analyzer is a common tool to analyze hardware performance counter data among many IBM eServer platforms, which includes systems running on AIX, i5OS, zOS, Linux on POWER, Linux on CELL/B.E.. Counter Analyzer accepts hardware performance counter data generated by AIX tools hpmc and tcount in the form of a cross-platform XML file format. The tool uses either build-in hsqldb database engine or external DB2 instance to store the raw performance counter data. The tool provides multiple views to help users identify and eliminate performance bottlenecks by examine the hardware performance counter values, computed performance metrics and also CPI breakdown models.

Trace Analyzer visualizes Cell/B.E. traces containing information such as DMA communication, locking/unlocking activities, mailbox messages, etc. Trace Analyzer shows this data organized by core, along a common timeline. Extra details are available for each kind of events, for example, lock identifier for lock operations, accessed address for DMA transfers, etc.

Visual Performance Analyzer is available as an IBM internal use tool for any IBMer who wants to try it out. Support is provided on a best-effort basis.

How does it work?

Profile Analyzer parses system profiles into an internal profiling data model that supports the profile hierarchy, offset locations, tick counts, CPU counter data, source line information, and disassembly. The plug-in then displays this data model, using various Eclipse views. The system profiles are those produced by Performance Inspector and AIX® Tprof. However, Visual Performance Analyzer can be extended to support almost any platform by converting a system profile to an XML schema that it understands.

Code Analyzer is able to read profiling information generated by AIX Tprof or FDPR-Pro performance tools. It reads in executable files and shared libraries and analyzes them using FDPR-Pro. FDPR-Pro is a post-link analyzer and performance optimization tool that can perform accurate static and dynamic analysis of executable files.

Pipeline Analyzer reads the .pipe and .config input files that are produced by the IBM Performance Simulator for Linux on POWER. An instruction trace is first collected and analyzed by a processor model. The two output files are produced for viewing with either the Performance Simulator or Visual Performance Analyzer.

Counter Analyzer reads the counter data files as its input, parses these files into an internal counter data model, and then displays this data model using various Eclipse views. The counter data files are generated by hpmc or tcount, having a suffix ".pmf".

Trace Analyzer reads in traces generated by the Performance Debugging Tool for Cell, and displays time-based graphical visualization of the program execution as well as a list of trace contents and the event details for selection.

1.1VPA on Alphaworks

Visual Performance Analyzer was released on Alpha works to explore the use of Eclipse-based performance tools with IBM customers. VPA is built as an Eclipse Rich Client Platform (RCP) package and there are versions for AIX and Windows. An RCP release contains Eclipse runtime files, all required plug-ins and VPA plug-ins.

1.2Release History

Date	Description
09/14/2006	Initial release of VPA to Alphaworks
06/08/2007	VPA 5.0 Release

2.VPA Basics

Visual Performance Analyzer is an Eclipse-based tool set that includes: Profile Analyzer, Code Analyzer, Pipeline Analyzer, Counter Analyzer, and Trace Analyzer. All of these tools are Eclipse plug-ins.

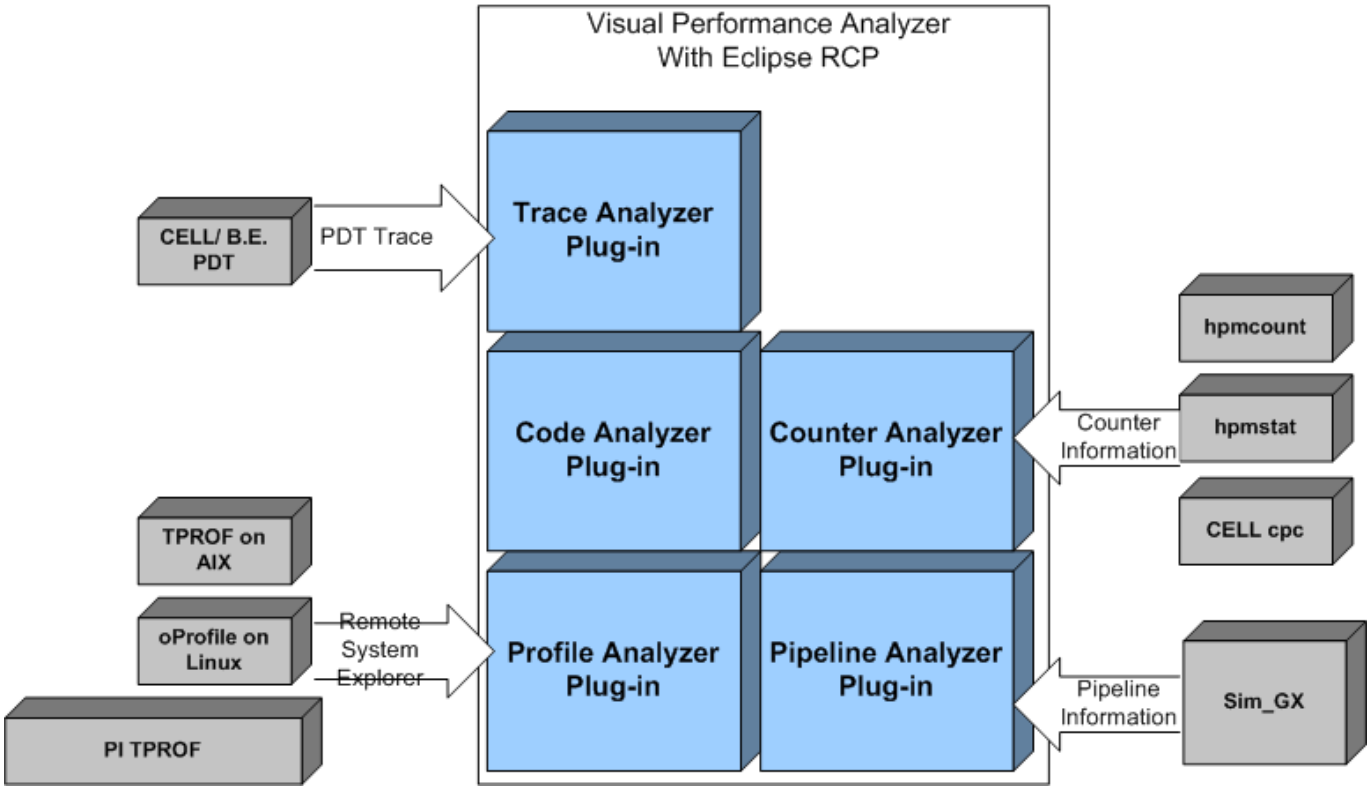


Figure 1 System Architecture of Visual Performance Analyzer

Profile Analyzer

Profile Analyzer is a system profile analysis tool. This plug-in obtains profile information from various platform specific tools, and provide analysis views for user to identify performance bottle necks.

Pipeline Analyzer

Pipeline Analyzer provides almost the same features as ScrollPipeViewer, a standalone Java application. It gets pipeline information of Power processors from the Sim-GX tool, and provides two analysis views; scroll mode and resource mode.

Code Analyzer

Code Analyzer reads XCOFF (AIX binary file format) files or ELF files running on Linux on Power, and displays program structure with block information. With related profile information, it can provide analysis views on hottest program block as well as some optimization suggestions.

Counter Analyzer

Counter Analyzer reads counter data files generated by hpmc or tcount running on AIX, and it provides multiple views to help users identify and eliminate performance bottlenecks by examine the hardware performance counter values, computed performance metrics and also CPI breakdown models.

Trace Analyzer

Trace Analyzer reads in traces generated by the Performance Debugging Tool for Cell, and displays time-based graphical visualization of the program execution as well as a list of trace contents and the event details for selection.

2.1Design Objectives

The base object of Visual Performance Analyzer is to extend the capabilities of Eclipse by adding plug-in support for: system profile, code, pipeline, and counter analysis. VPA is a collection of performance data analysis tools that can be used to identify performance bottlenecks. VPA does not supply performance data collection tools. Instead, it relies on platform specific tools, like AIX Tprof, to collect the performance data. When necessary, multi-platform support is provided by converting data into XML. The XML schema is understood by VPA and is parsed and loaded for analysis. The VPA tool must be extensible and it achieves this by allowing for additional plug-ins to be added and also by adding integration between plug-ins, e.g. shared internal data models and linked views.

Information about VPA data files:

- The .etm is the XML file for Profile Analyzer
- .etz is the zipped XML profile data
- .opm is the XML file for Profile Analyzer
- .opz is the zipped XML file for Profile Analyzer
- Java profile data from IBM JRE Java profiling tools are merged by TProf tools into a single .etm file. No additional post processing is needed.
- The pipeline files are: .pipe data file and .config file is the default configuration file.
- The .pmf is the XML file for Counter Analyzer.
- The .pe is the file for Trace Analyzer

2.2Deployment

As a performance analysis tool, Visual Performance Analyzer typically runs on User's ThinkPad or desktop as a client application. Visual Performance Analyzer can get performance-related data from servers via Remote System Explorer (RSE) or by copying the files with FTP or some other means.

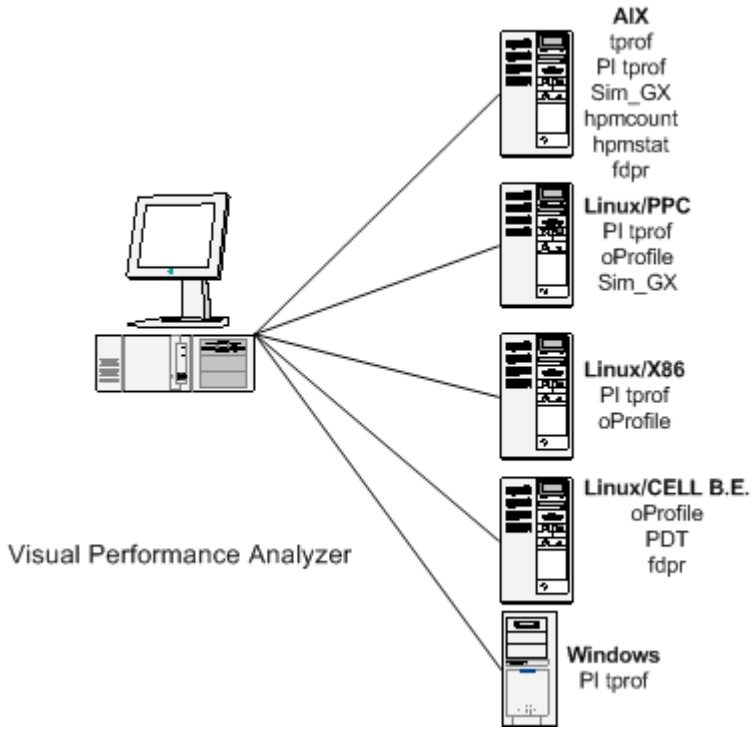


Figure 2 System Deployment of Visual Performance Analyzer

2.3Software Stack Information

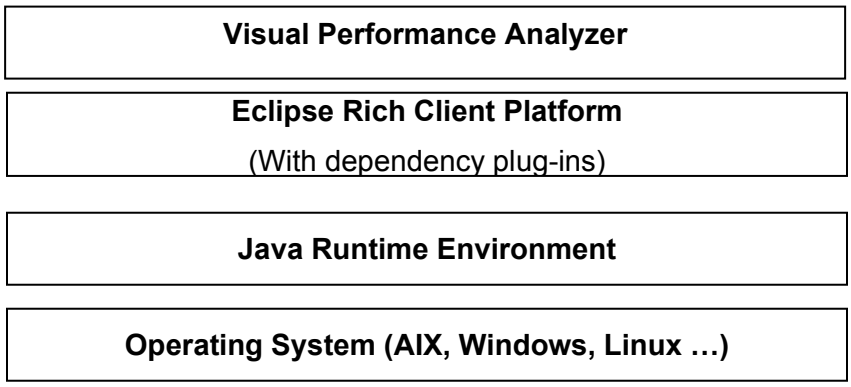


Figure 3 Product Stack of Visual Performance Analyzer

VPA runs on the following Operating Systems:

(1) Windows XP with SP2 or later

(2) IBM AIX 5.3 in latest Maintenance Level

(3) Linux/x86--Fedora Core 5/6

Profile Analyzer, Pipeline Analyzer, Trace Analyzer and Counter Analyzer are Eclipse plug-ins and are 100% JAVA code. They can run on all above supported platforms.

Code Analyzer is also an Eclipse plug-in, but it depends on FDFR-Pro libraries that are platform-dependent libraries. Code Analyzer can only run on Windows in this release.

Although VPA only runs on the above operating systems, it's important to realize that it can analyze data collected from any platform, providing the data is provided in a format understood by VPA.

VPA supports only IBM J9 JRE 5.

There is an IBM J9 JRE5 in VPA

VPA supports the following Eclipse platforms:

(1) Eclipse 3.2 with latest fixpack, such as Eclipse 3.2.2

3.Installation

No installer is required for VPA installation. The VPA installation is as simple as:

1. Download a newest VPA RCP release, usually it should be a zip archive or compressed tar archive
2. Extract the archive
3. Run the application by executing the Eclipse launcher script.

The RCP application will not include the following products: Performance Inspector for Windows and DB2 UDB. If you want to use these capabilities, they must install the corresponding product manually.

Configuration

No configuration is required for the VPA application installation.

Advance configuration information is provided in online-help. These configurations address some special requirements, such as setting bigger heap size of JVM for Eclipse when a user analyzes large profile.

Uninstallation

No special uninstallation action is required. If a user wants to uninstall a VPA RCP application, they can simply delete the application directory that VPA was installed to.

3.1Windows

These steps will walk you through the installation of VPA on your Windows workstation.

3.1.1Download from Alpha works

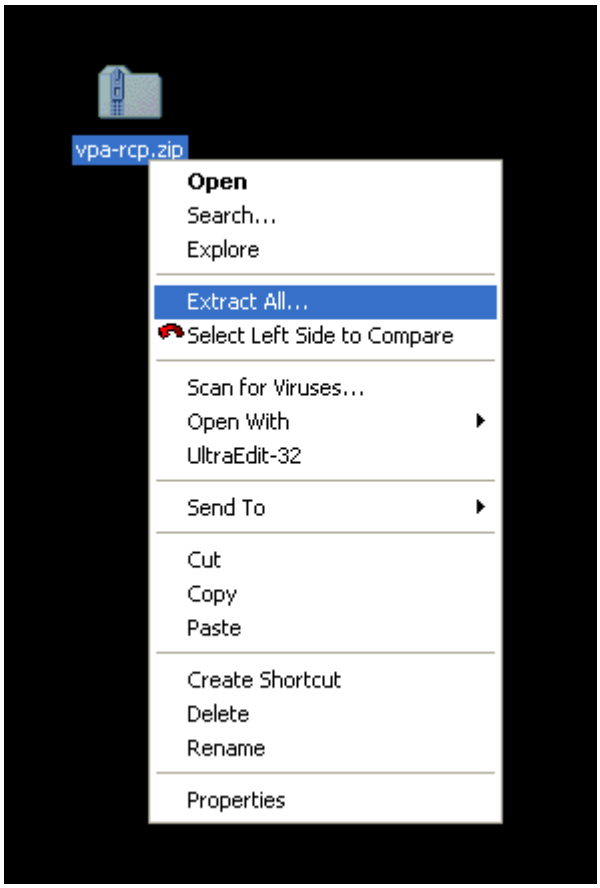
Download the latest VPA (Visual Performance Analyzer) from here:

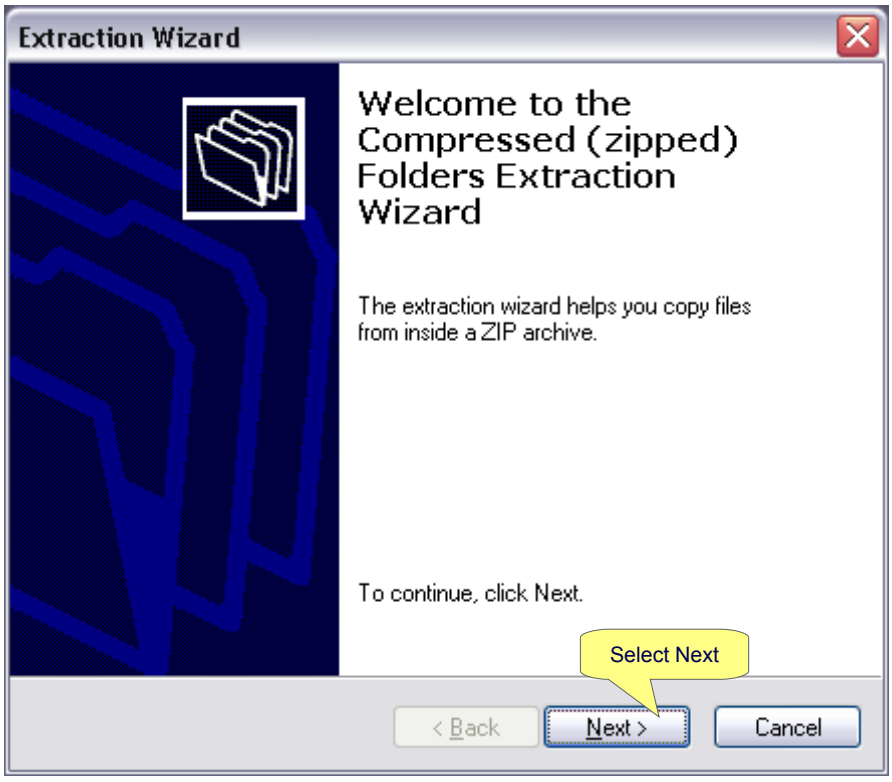
http://www.alphaworks.ibm.com/tech/vpa

Save vpa-rcp-${version}-win32.zip to your favorite download directory.

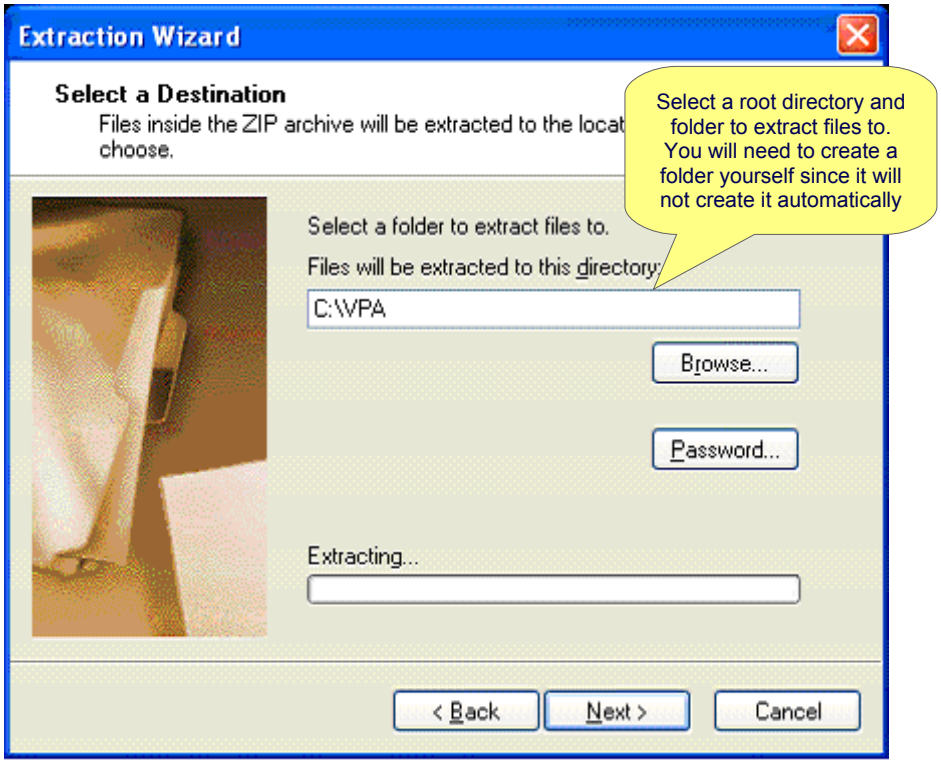
3.1.2Extract the compressed file.

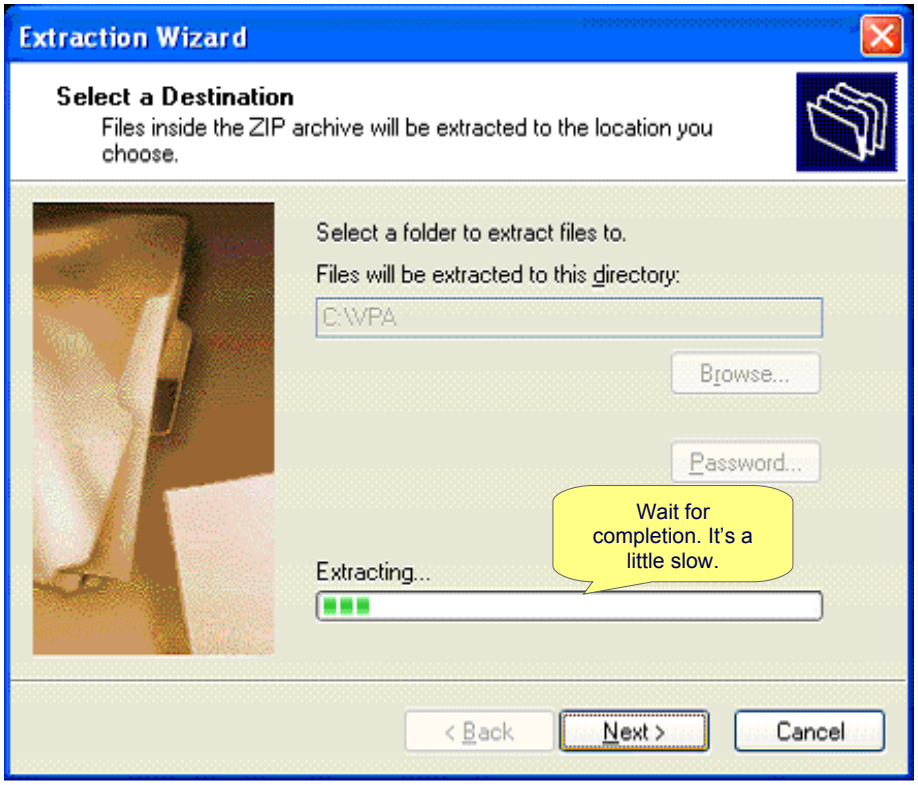
Right Click on the file and select Extract All to open the Extraction Wizard.

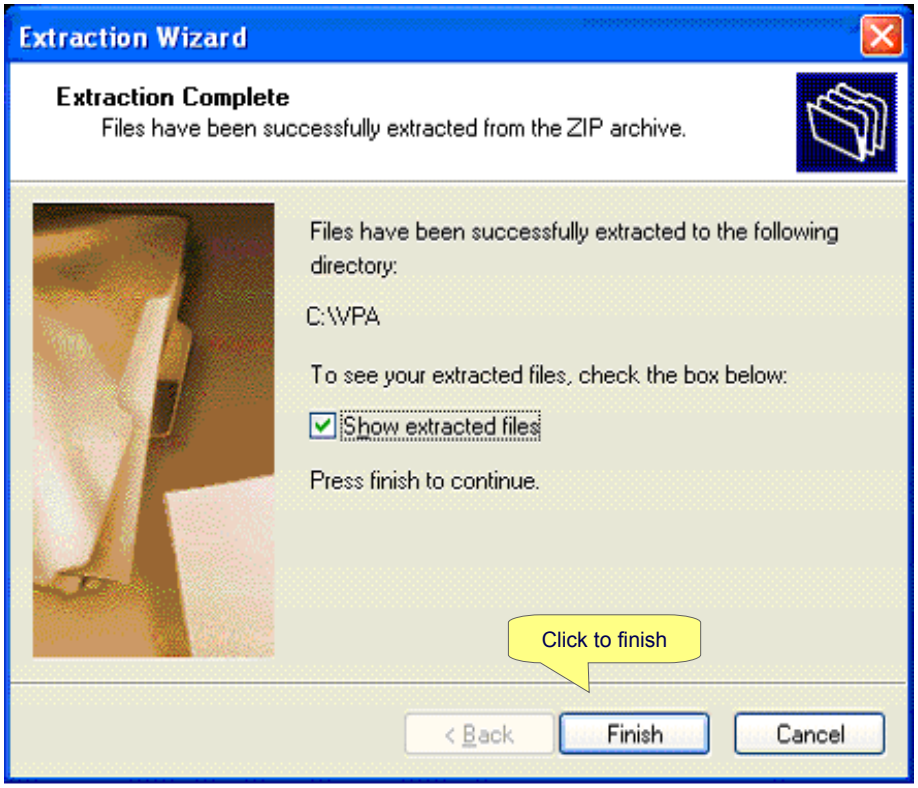

vpa-rcp.zip
Open
Search...
Explore
Extract All...
Select Left Side to Compare
Scan for Viruses...
Open With
UltraEdit-32
Send To
Cut
Copy
Paste
Create Shortcut
Delete
Rename
Properties


Extraction Wizard
Welcome to the Compressed (zipped) Folders Extraction Wizard
The extraction wizard helps you copy files from inside a ZIP archive.
To continue, click Next.
Select Next
< Back
Next >
Cancel

As time advances, new versions of VPA will come out. In order to save yourself a lot of headaches with new versions, create the new folder with a name containing the version number and install VPA to that directory. If each version is installed this way, you'll have multiple working versions. When there is a problem, you can go back to the old version.

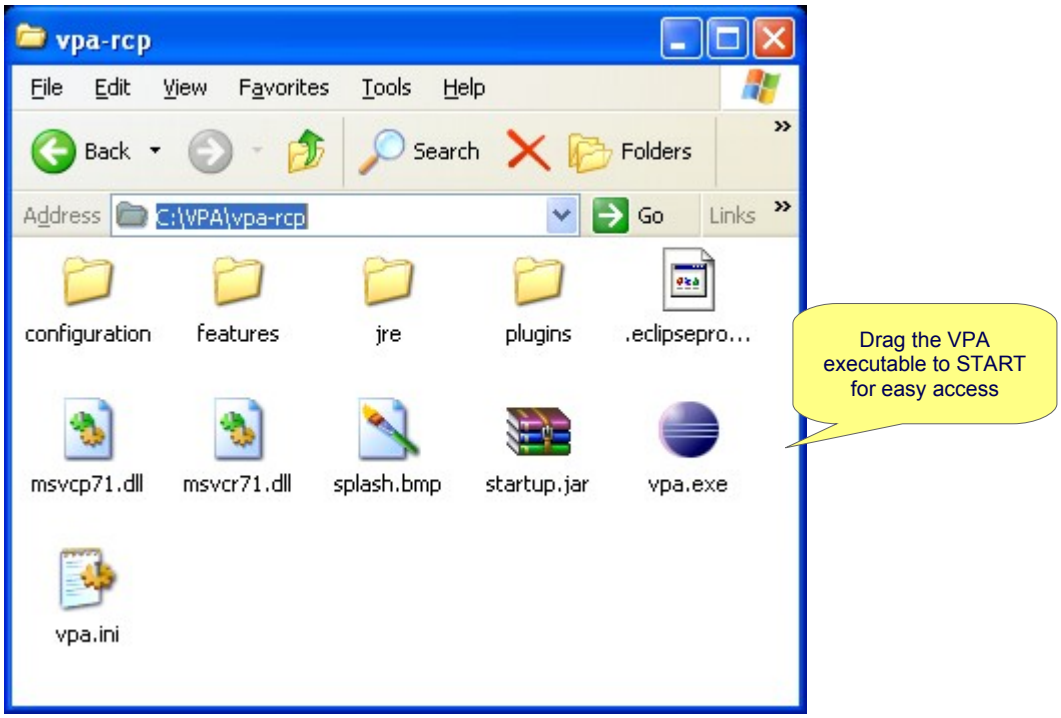



Extraction Wizard
Select a Destination
Files inside the ZIP archive will be extracted to the location you choose.
Select a folder to extract files to.
Files will be extracted to this directory:
C:\VPA
Browse...
Password...
Wait for completion. It's a little slow.
Extracting...
< Back
Next >
Cancel

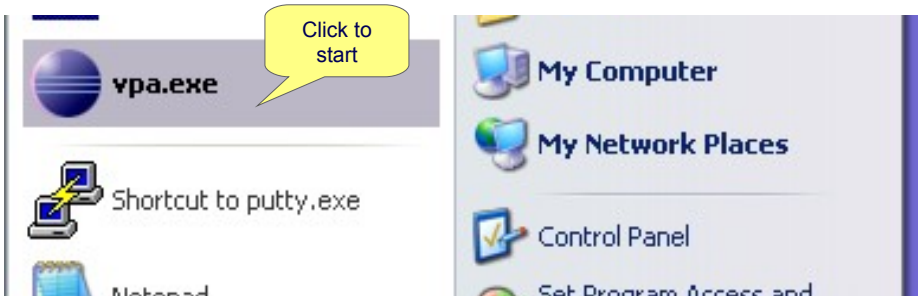

Extraction Wizard
Extraction Complete
Files have been successfully extracted from the ZIP archive.
Files have been successfully extracted to the following directory:
C:\VPA
To see your extracted files, check the box below:
Show extracted files
Press finish to continue.
Click to finish
< Back
Finish
Cancel

3.1.3Create a Shortcut

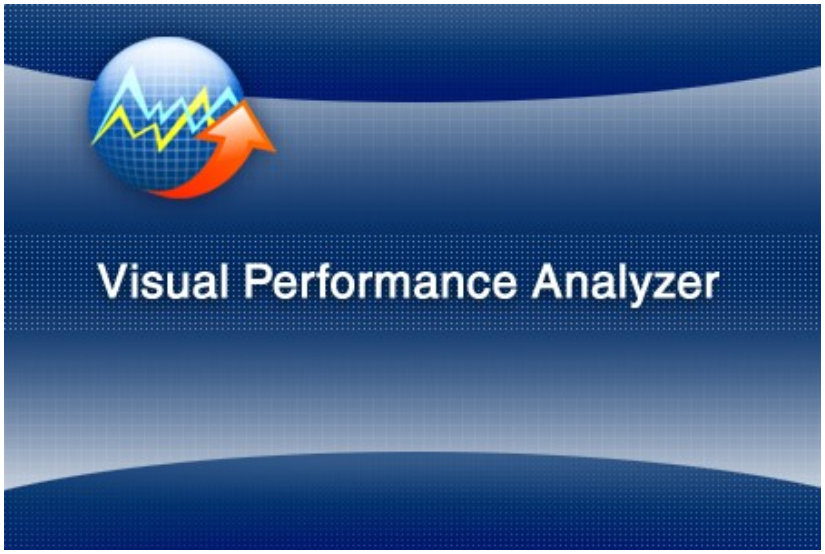
A window with the folder and its contents will open if you selected “Show extracted files”.



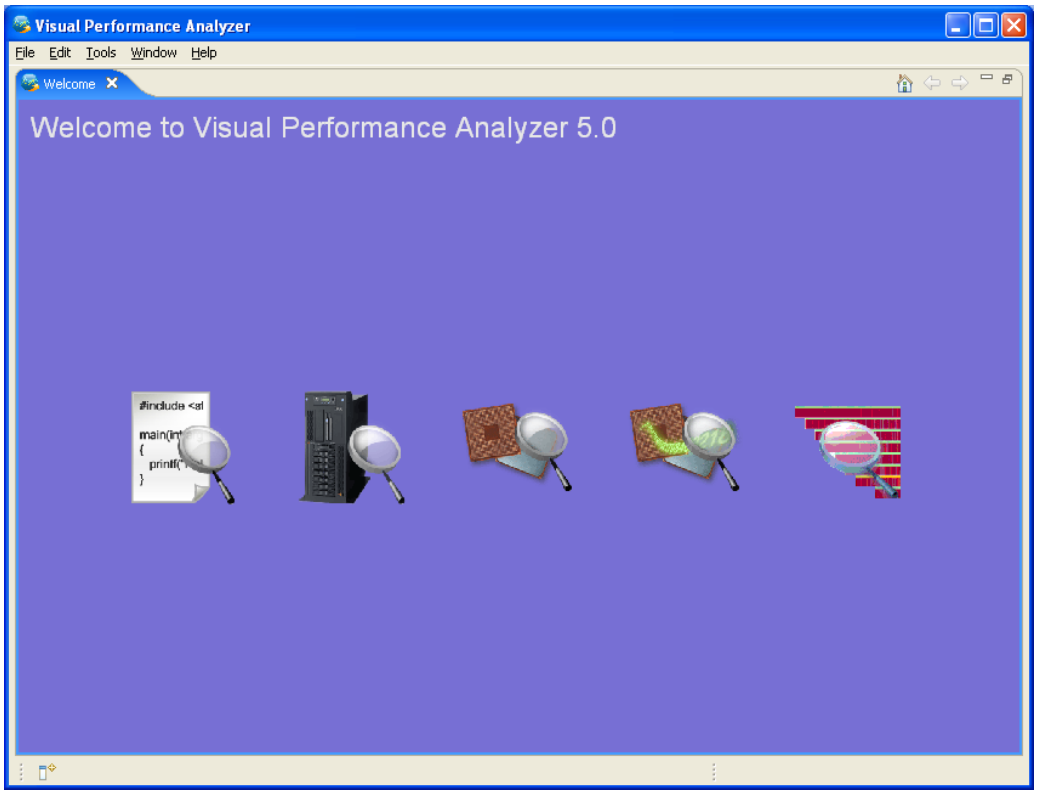
Click on the VPA executable to start VPA.



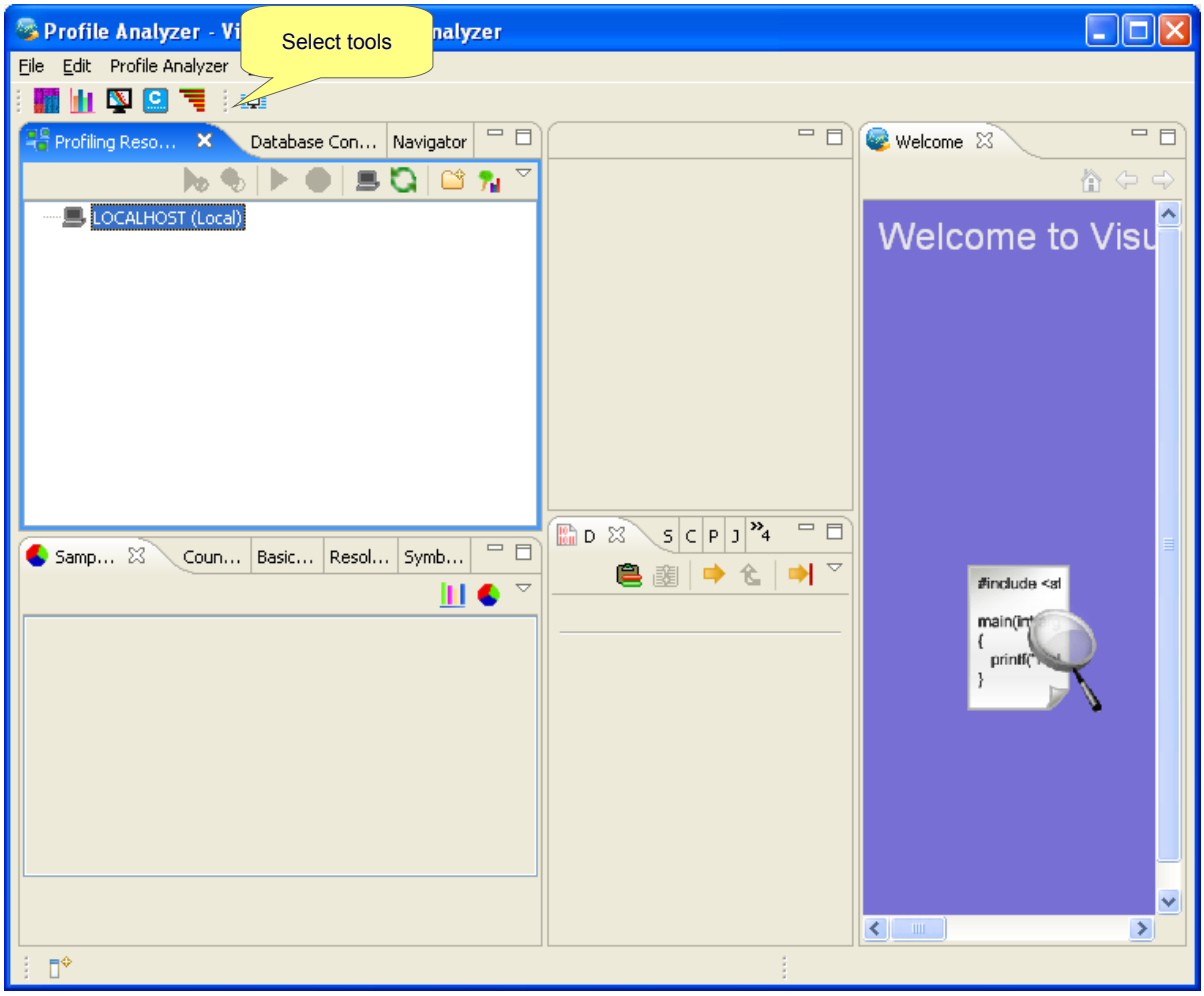
Otherwise you will see this:

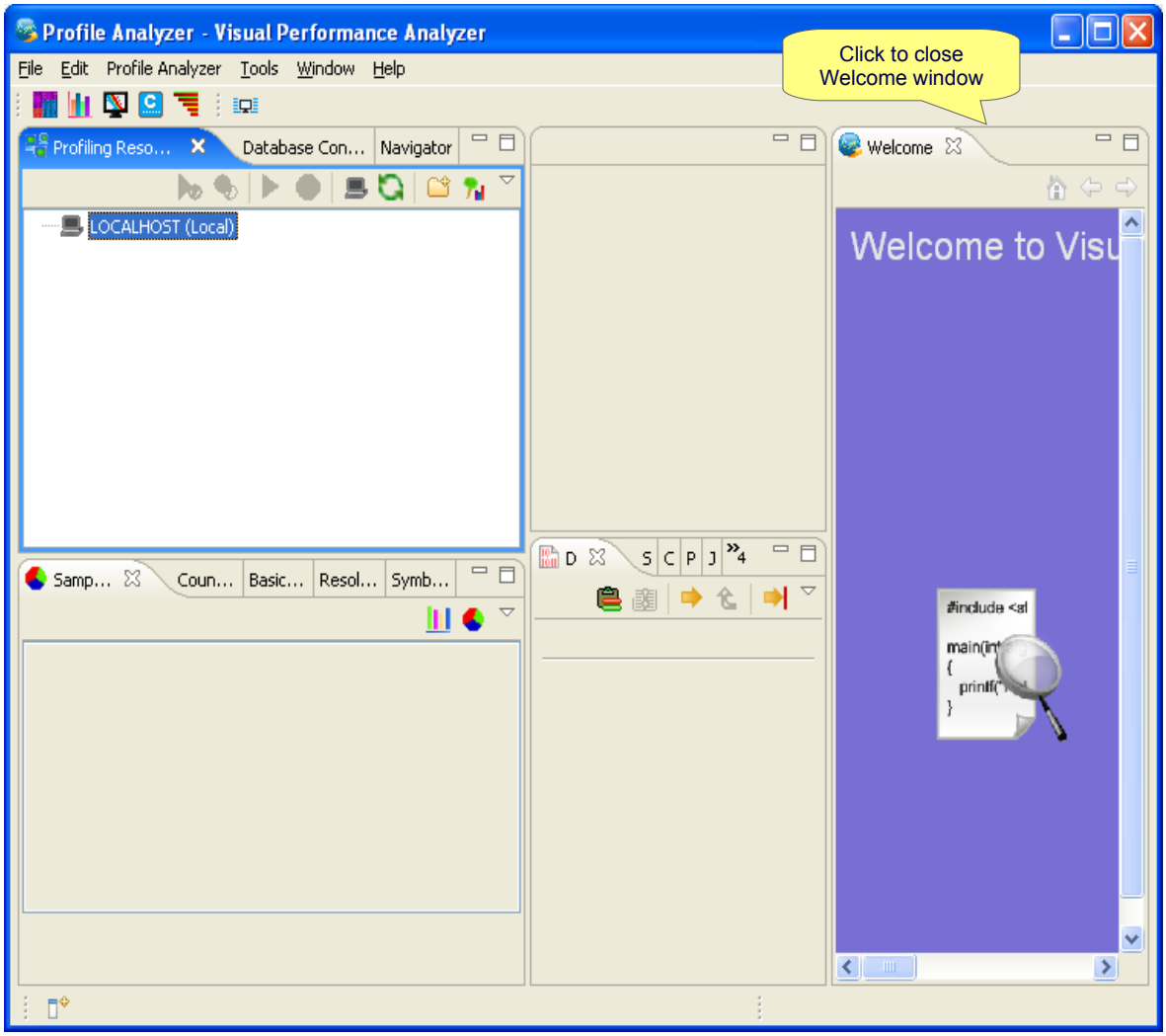


If you see this screen when you start up VPA …

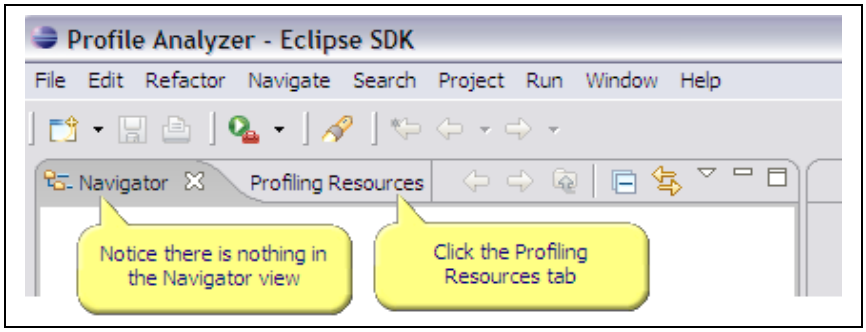


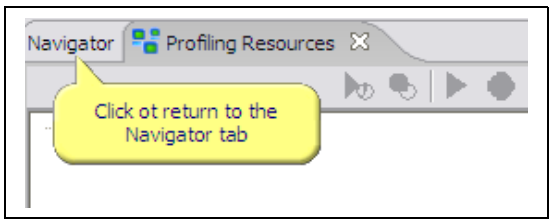
It means that Eclipse is not running one of the VPA tools and you will need to switch to one of the VPA tool perspective by following this procedure.

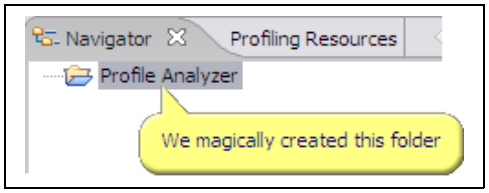

Select tools
File Edit Profile Analyzer
Profiling Reso...
Database Con...
Navigator
LOCALHOST (Local)
Welcome
Welcome to Visu
Samp...
Coun...
Basic...
Resol...
Symb...

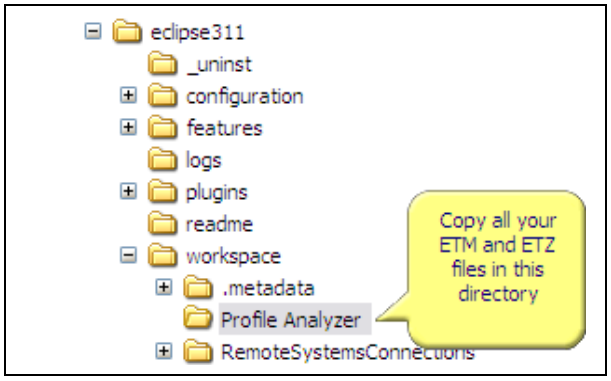

Profile Analyzer - Visual Performance Analyzer
File Edit Profile Analyzer Tools Window Help
Click to close Welcome window
Profiling Reso...
Database Con...
Navigator
LOCALHOST (Local)
Welcome
Welcome to Visu
Samp...
Coun...
Basic...
Resol...
Symb...
D
S C P J

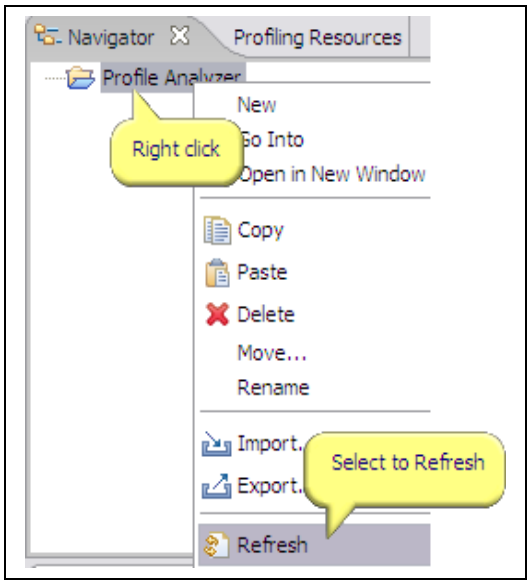
If you have existing profiles, and do not see them after a VPA upgrade, follow these steps to restore them to the project.

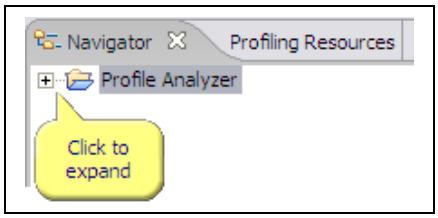


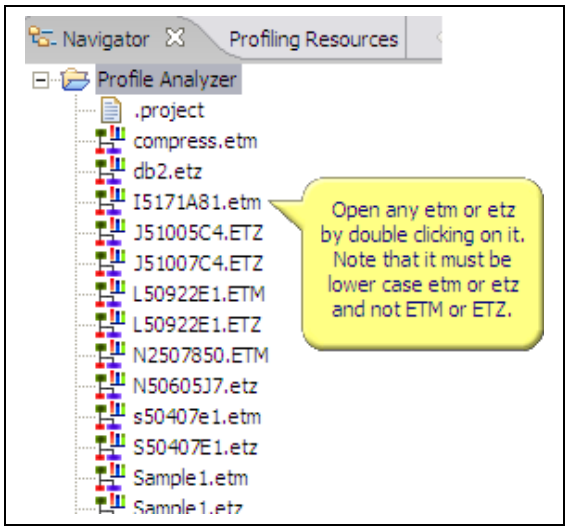












3.2Linux

These steps will walk you through the installation of VPA on your Linux workstation.

Supported Linux platforms are: Linux/x86: Fedora Core 5/6

3.2.1Download from Alphaworks

Download the latest VPA (Visual Performance Analyzer) from here:

http://www.alphaworks.ibm.com/tech/vpa

3.2.2Extract the compressed file

Go to the directory where gz file is .....cd /favdir
Change file attributes .....chmod 755 vpa-rcp-${version}-linux-x86.tgz
Decompress the file .....tar –xvfz vpa-rcp-${version}-linux-x86.tgz

---

3.3AIX

These steps will walk you through the installation of VPA on your AIX workstation.

3.3.1Download from Alphaworks

Download the latest VPA (Visual Performance Analyzer) from here:

http://www.alphaworks.ibm.com/tech/vpa

Save vpa-rcp-${version}-aix-ppc.zip to your favorite download directory.

3.3.2Extract the compressed file

Go to your favorite download directory and follow these steps to extract the VPA tool:

Go to the directory where gz file is ……cd /favdir
Change file attributes ……chmod 755 vpa-rcp-${version}-aix-ppc.tgz
Decompress the file ……gzip –dc vpa-rcp-${version}-aix-ppc.tgz | tar xvf –

4.Collecting Performance Data

VPA is a collection of performance data analysis tools. It relies on platforms to provide the necessary tools for collecting data and converting the data into a format that is understood by VPA.

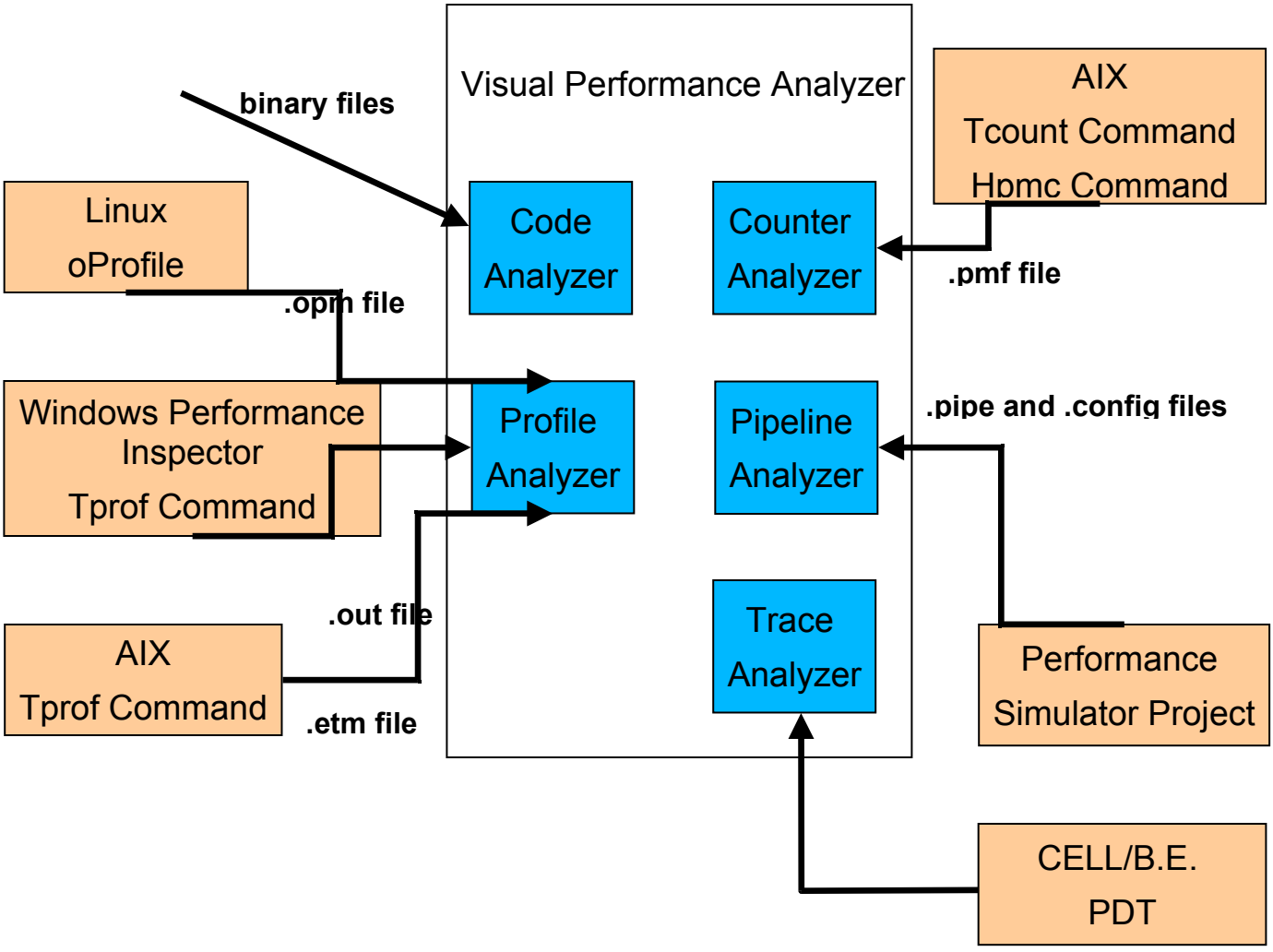
4.1Using Platform Tools

Visual Performance Analyzer works with following tools for collecting profile data.

- AIX Tprof
- Performance Inspector for Windows Tprof
- IBM JRE Java profiling tools
- Linux oProfile

Profile data from AIX tprof is converted into XML by using the –X flag. The .etm is the XML file for Profile Analyzer; .etz is the zipped XML profile data. Profile data from PI Tprof is in a .out format, which profile analyzer supports directly. Java profile data form IBM JRE Java profiling tools are merged to above tools.

Pipeline data is generated from tools found in the IBM Performance Simulator for Linux on POWER™ project on Alphaworks. The .pipe file is pipeline data file and .config file is the default configuration file.



4.2Setting up Windows to collect Profiling data

4.2.1Verify that your Java Runtime is installed on your system

Run the following command:

java –version

You should see something similar to the following:

java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142-20050609 (JIT enabled: jitc))

Note: You need version 1.4.1 or higher

4.2.2Verify that the Windows performance tools are installed

VPA runs with the Performance Inspector for Windows performance tools. Run the following command:

Swtrace -?

You should see something similar to the following:

D:\>swtrace -?

SWTRACE Version: 7.1.1

Valid SWTRACE commands: …

The Performance Inspector for Windows package can be downloaded from here:

http://www.alphaworks.ibm.com/tech/pi

4.2.3Verify PI Tprof

Using the PI tools themselves, you can verify their operation by capturing a system trace using these steps:

Swtrace init

Swtrace enable Tprof

Swtrace on

Swtrace off

Swtrace get

Swtrace post

Post

At this point you should have a PI profile (.out file) in your working directory that you can look at. Refer to PI documentation for details on PI tools.

You can capture traces yourself or you can configure VPA to collect traces. Refer to the Profile Analyzer plug-in section in this document.

4.2.4Copying data files

Running Performance Inspector for Windows Tprof, produces an ascii profile (.out) file. You can simply use FTP to transfer the file to your system running VPA or open the profile locally if you have VPA installed on the same system. See section 4.4 about using Remote System Explorer.

4.3Setup up AIX to collect Profiling data

4.3.1Verify that your Java Runtime is installed on your system

Run the following command:

java –version

You should see something similar to the following:

java version "1.4.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1)
Classic VM (build 1.4.1, J2RE 1.4.1 IBM build cxppc321411-20040301 (JIT enabled: jitc))

Note: You need version 1.4.1 or higher

4.3.2Verify that the AIX performance tools are installed

Recent versions of AIX Tprof can generate XML profiles. AIX 5.3.TL5 or higher is required. The utility that produces a VPA profile from the Tprof output is bundled with the bos.perf.tools package. It includes an updated versions of Tprof, Symlib and the added tprof2xml utility.

Verify installation of bos.perf.tools package:

lslpp –L bos.perf.tools | grep “bos.perf”

If not installed, you can use smitty or installp

4.3.3Verify AIX Tprof

Using the AIX tools, you can verify their operation by capturing a system trace using these steps:

tprof -eukj -X -A -F -r vpa_test -x sleep 5

At this point you should have a Tprof profile (vpa_test.etm file) in your working directory that you can look at.

You can capture traces yourself or you can configure VPA to collect traces. Refer to the Profile Analyzer plug-in section in this document.

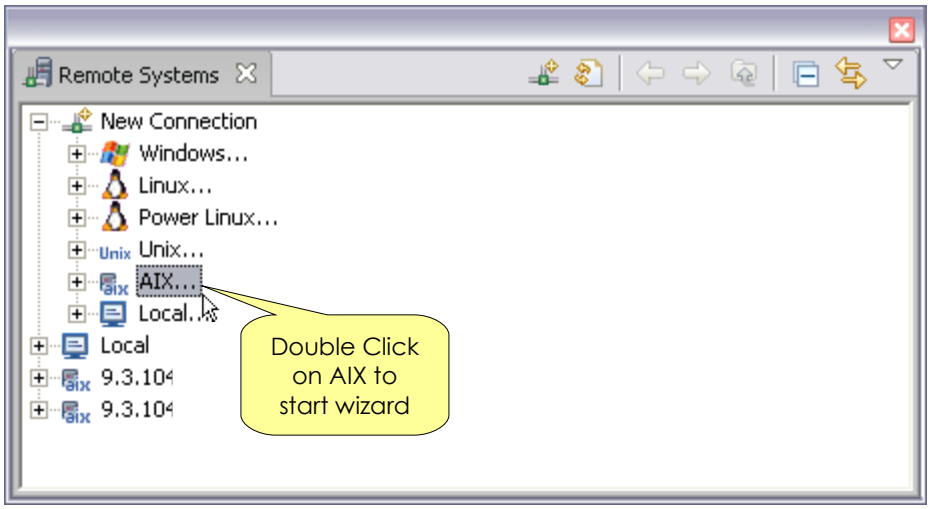
4.3.4Copying data files

All versions of AIX support FTP. So, once AIX Tprof has produced the XML profile (.etm) file, you can simply use FTP to transfer the file to your system running VPA. If VPA has been installed on the same AIX system you can open the profile locally. See section 4.3.5 about using Remote System Explorer.

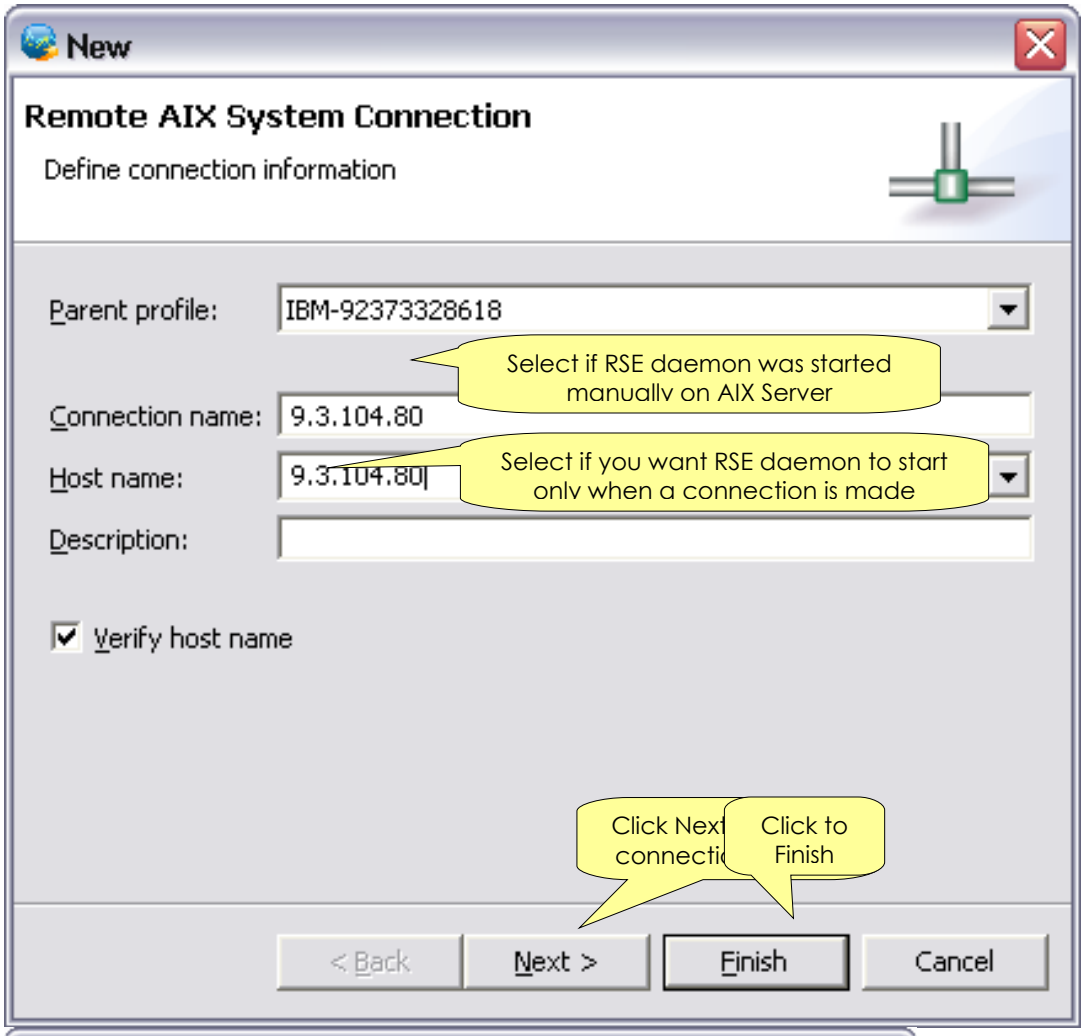
4.3.5Using Remote System Explorer

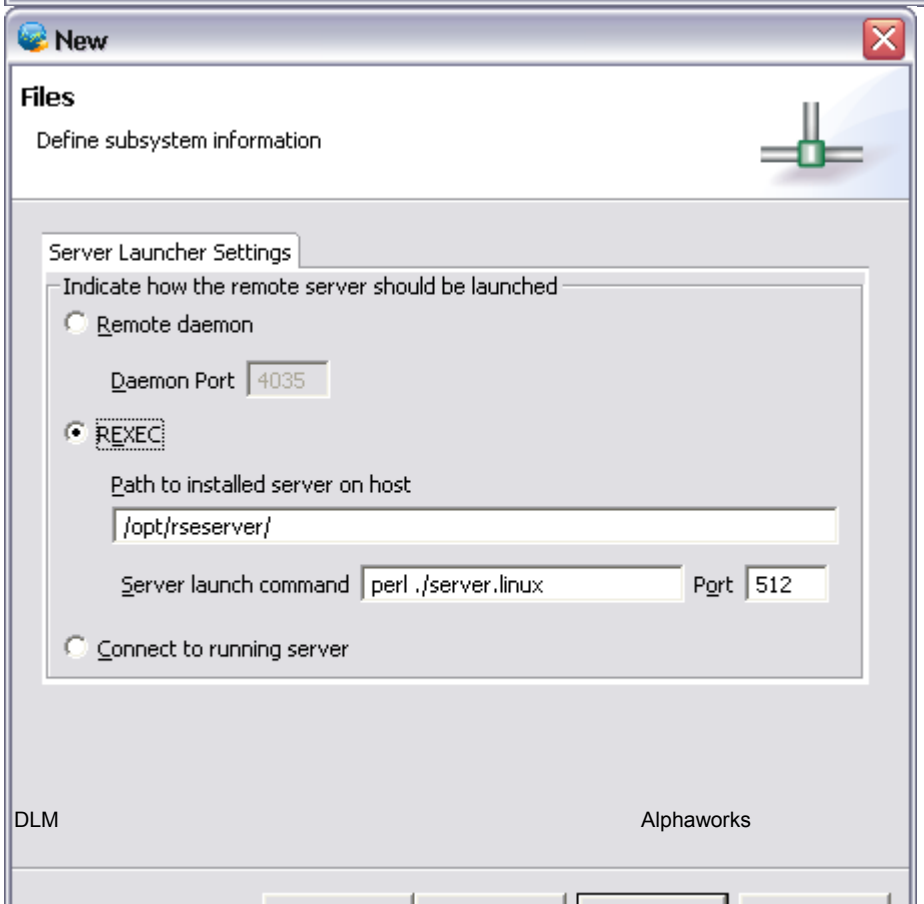
You can configure VPA to use Remote System Explorer (RSE) to remotely collect data and transfer files. However, VPA does not distribute an RSE server component. The below steps illustrate how to configure an AIX remote resource but the steps are similar to configuring a Windows remote resource as well.

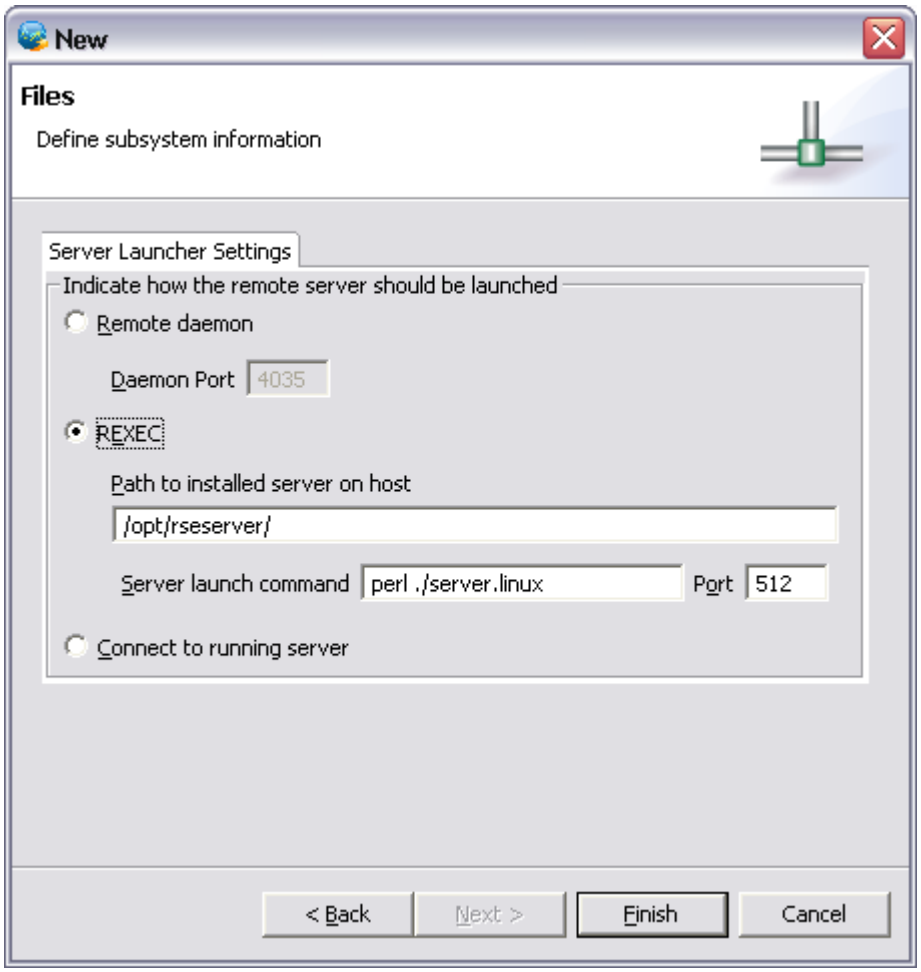
Open Remote Connection View ……	Choose Window → Show view → Other → Remote Systems → Remote Systems
Open Remote AIX System Connection wizard …	Under Remote Systems window, **double click** New Connection → AIX



Follow wizard to specify information about the remote System


New
Remote AIX System Connection
Define connection information
Parent profile:
IBM-92373328618
Select if RSE daemon was started manually on AIX Server
Connection name:
9.3.104.80
Select if you want RSE daemon to start only when a connection is made
Host name:
9.3.104.80
Description:
Verify host name
Click Next
connecti
Click to Finish
< Back
Next >
Finish
Cancel


New
Files
Define subsystem information
Server Launcher Settings
Indicate how the remote server should be launched
Remote daemon
Daemon Port
4035
REXEC
Path to installed server on host
/opt/rseserver/
Server launch command
perl ./server.linux
Port
512
Connect to running server



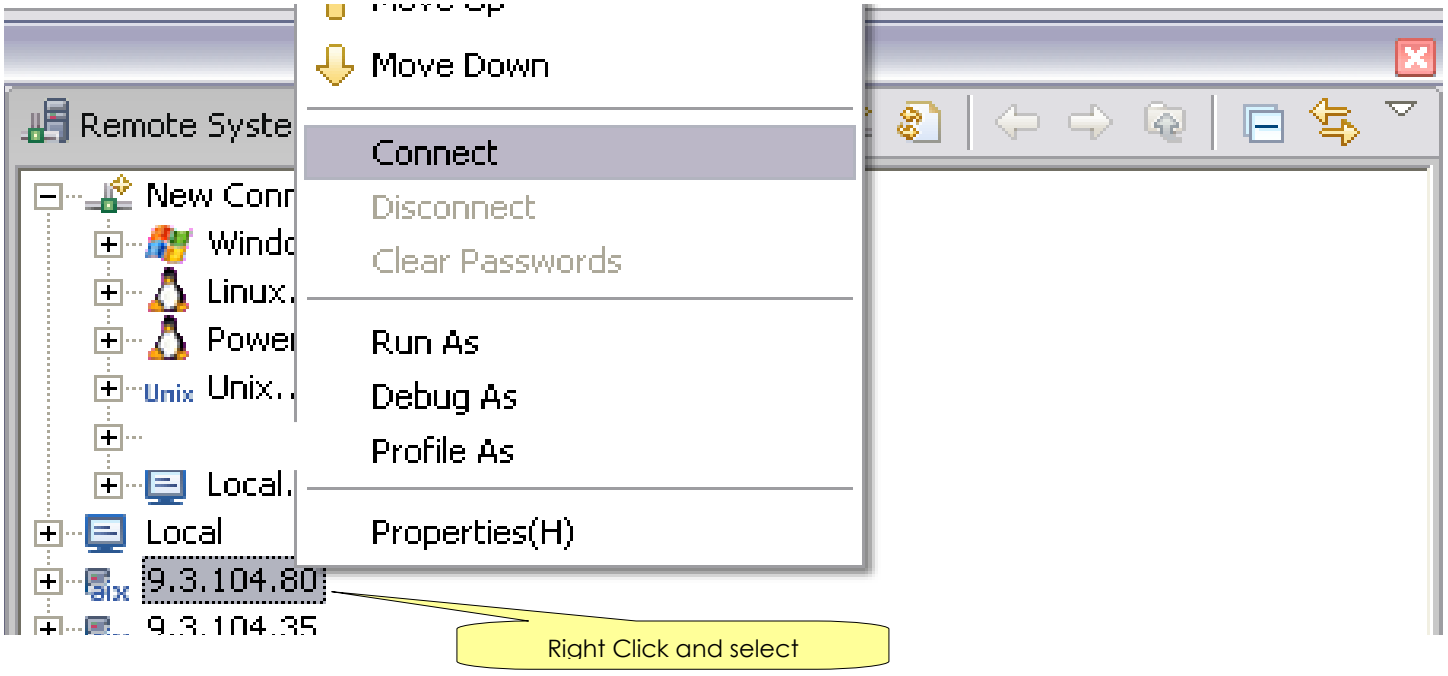
Note:

If RSE daemon was started manually on the AIX server choose Remote daemon option. The port 4035 is selected by default.

If you want RSE daemon to be started automatically when a connection is made, select REXEC option and specify where the server launch command is found. This is typically a perl script.

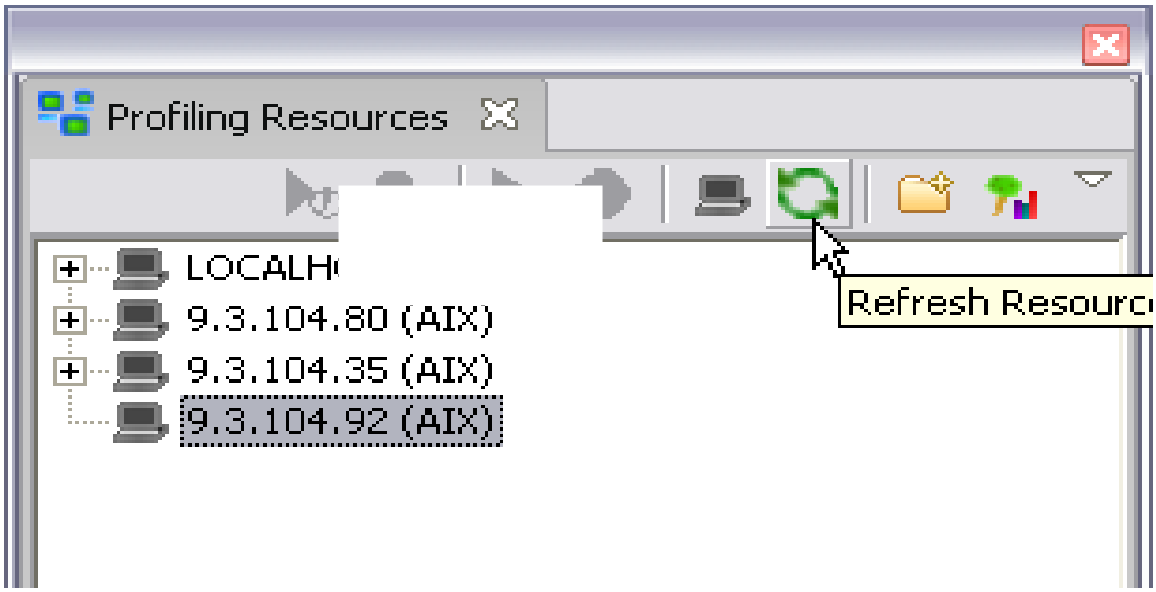
Connect to remote server .....

Right click the new connection and select Connect



Type username / password

Go to Profiling Resources, click refresh to see the new connection



4.4Collecting Profiling Data on Linux platform

4.4.1Linux CELL/B.E.

- Hardware: CELL/B.E. blade
- Software Requirement: fedora core 6 and CELL/B.E. SDK 2.0 installation
- Verify oprofile: opcontrol/opreport –X
- Tool Usage:

After verifying that oprofile has been installed successfully, users should first use “opcontrol --init” to initialize oprofile module; and then, use “opcontrol –event=event:count” to add an event to measure for the hardware performance counters (users can refer to event names and minimal counters by using “opcontrol –l”).

Next, use “opcontrol --separate=all” to separate samples based on the given separator. It is not an optional step, user must process it to meet VPA requirement.

Users can use “opcontrol --start” to start collecting profiling data, and start one user application: then, use “opcontrol --stop” to stop collecting profiling data; and use “opcontrol --dump” to force a flush of the collected profiling data to the daemon; Finally, use “opreport -X –g –l –d –o xxx.opm” to generate a specified XML output, which can be imported to Profile Analyzer. The xml output file must be suffixed with the extension ‘.opm’, which identifies an acceptable file.

Users can further use “opcontrol --reset” to clear out data, and choose “opcontrol --deinit” to unload the oprofile module.

Some important command usages :

opcontrol --init :

loads the oprofile module and oprofilefs

opcontrol –event=event_name:count:unit_mask:kernel-space_count:user-space_count :

choose an event with specified event_name, count, unit_mask, kernel-space counting, user-space counting. Here, the unit_mask, kernel-space counting, user-space counting are optional.

A default event can be specified with the command " *opcontrol –event="default"* ". Generally, the default event is the system timer of the OS and hardware.

opcontrol -l :

list event types and unit masks

opcontrol --start/--stop/--reset/--deinit :

start running the oprofile, stop oprofile, reset the profile data in default session. Unload oprofile module.

opreport -X –g –d –l xxx.opm :

Generate a specified XML output

(Here: -X : specifies the output file in XML format.

-g : show source file and line for each symbol.

-l : list per-symbol information instead of a binary image summary.

-d : show per-instruction details for all selected symbols.)

4.4.2Linux PowerPC

- Hardware: System p servers or POWER blade
- Software Requirement: Linux, oprofile (oprofile Download Link: http://oprofile.sourceforge.net)
- Verify oprofile: the same as CELL/B.E.
- Tool Usage: the same as CELL/B.E.

4.4.3Linux X86

- Hardware: X86 based machine
- Software Requirement: Linux, oprofile (oprofile Download Link: http://oprofile.sourceforge.net)
- Verify oprofile: the same as CELL/B.E.
- Tool Usage: the same as CELL/B.E.

4.5Collecting Pipeline data on PowerPC

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another alphaWorks technology. Please refer to the directions given by this project for collecting pipeline data. While VPA provides the Pipeline data analysis tool, the project provides the tools necessary for collecting and generating Pipeline data files.

4.6Collecting Counter Data on PowerPC

Before using Counter Analyzer to view and operate counter data, you should first prepare data from data source.

1. Our Counter Analyzer supports opening the counter data file generated by both **hpmc** and **tcount**. The following are two instances:
    - hpmc -s 0.1 -mv -G 1,2,3 -m output.pmf -x sleep 5
    - tcount -g 4,5,6,7,8 -X output.pmf sleep 5
2. Make sure that the counter data file has the suffix ".pmf".

5.Using the VPA analysis tools

This section describes the use of each plug-in. structure of the system by first focusing on some typical usage scenarios where various tasks are performed, then outlining the major components of the system and their interactions. You can find this information by selecting Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.1Profile Analyzer

Profile Analyzer is a tool that allows you to navigate through a system profile, looking for performance bottlenecks. It provides a powerful set of graphical and text-based views to allow users to narrow down performance problems to a particular process, thread, module, symbol, offset, instruction or source line. It supports profiles generated by Performance Inspector (tprof) and AIX tprof . It also merges IBM JRE Java profile data when it is merged into the above profiles. To load huge profile data files and reduce memory footprint, Profile Analyzer now uses database to cache profile files. The current version supports DB2 and an embedded database.

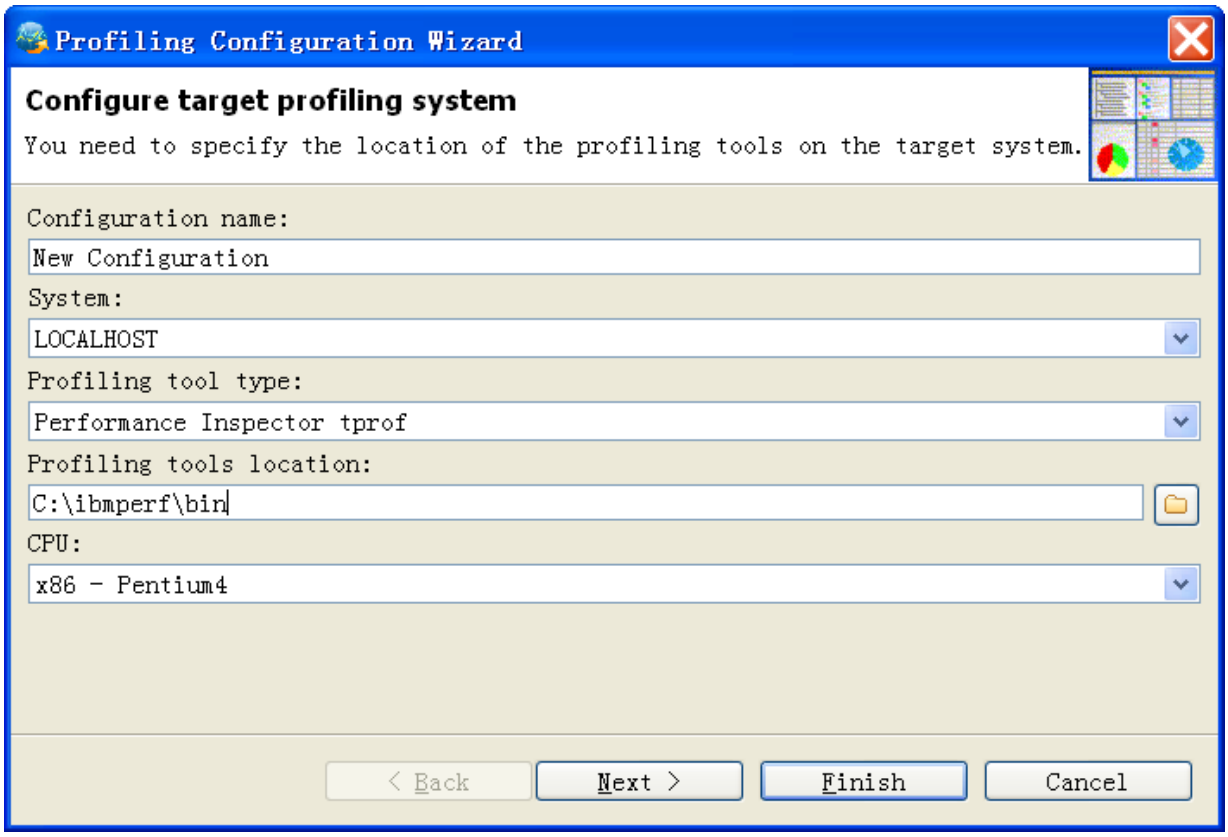
You can also find the Profile Analyzer User Guide from within VPA. Select **Help - Help Contents** within VPA. To get context sensitive help, press **F1** for Windows and AIX or press **Ctrl+F1** for Linux.

5.1.1Create a Profiling Configuration

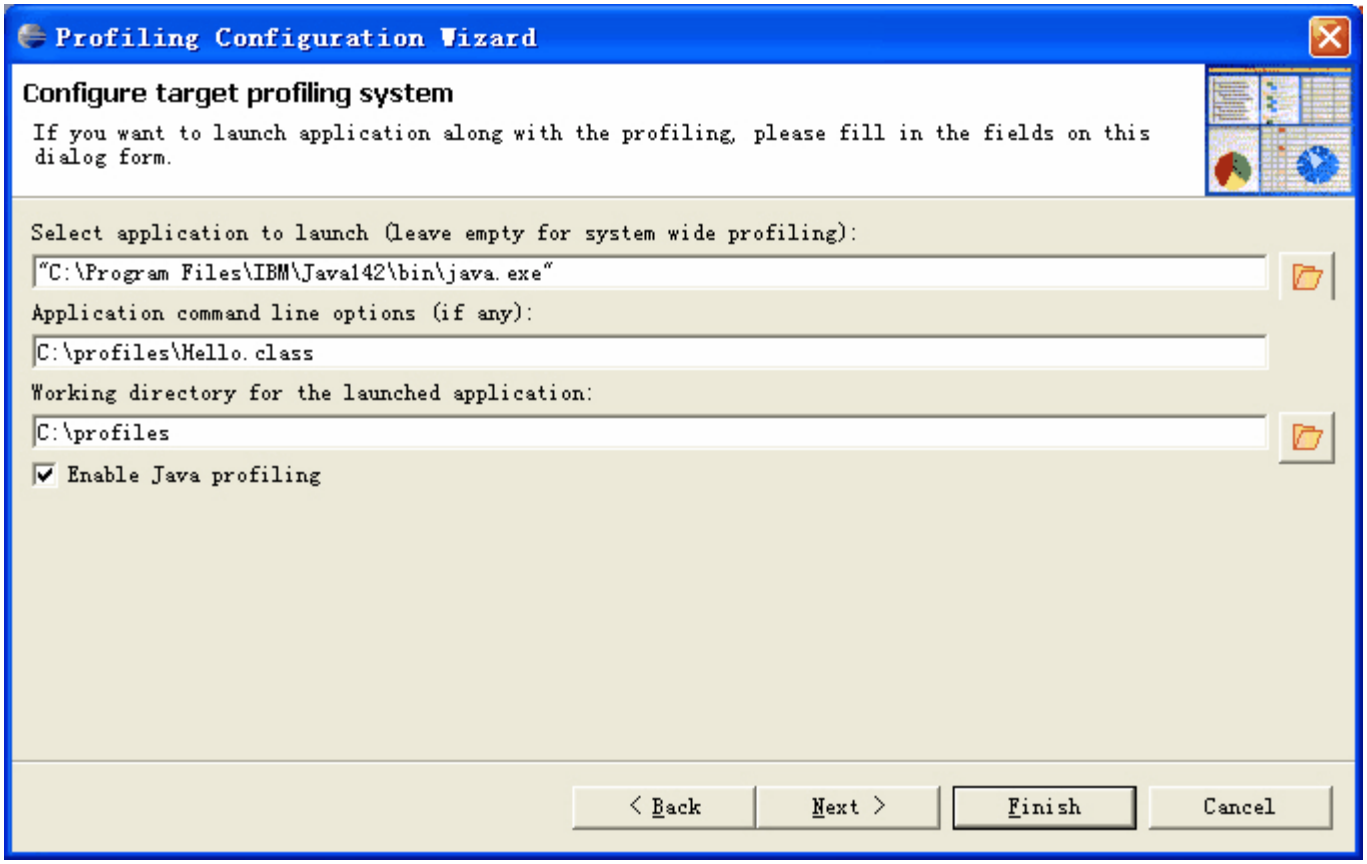
You can configure a system profile and have VPA run a workload and collect the profile. The steps are mostly the same for all supported systems. So, the example below will provide the steps necessary for creating a profiling configuration for a Windows system but are much the same as for any other system.

In the Profiling Resources view, right-click over a connection and choose **New Profiling Configuration**:

On the first page of the Profiling Configuration wizard, enter a name for the configuration (to remind you of what purpose the configuration serves).Currently, there are two kinds of tprof tools: Performance Inspector tprof and AIX tprof. Since you want to profile on Windows, you should choose Performance Inspector tprof and specify the profiling tools location. Then select the CPU type from the dropdown list. Click **Next**.


Profiling Configuration Wizard
Configure target profiling system
You need to specify the location of the profiling tools on the target system.
Configuration name:
New Configuration
System:
LOCALHOST
Profiling tool type:
Performance Inspector tprof
Profiling tools location:
C:\ibmperf\bin
CPU:
x86 - Pentium4
< Back
Next >
Finish
Cancel

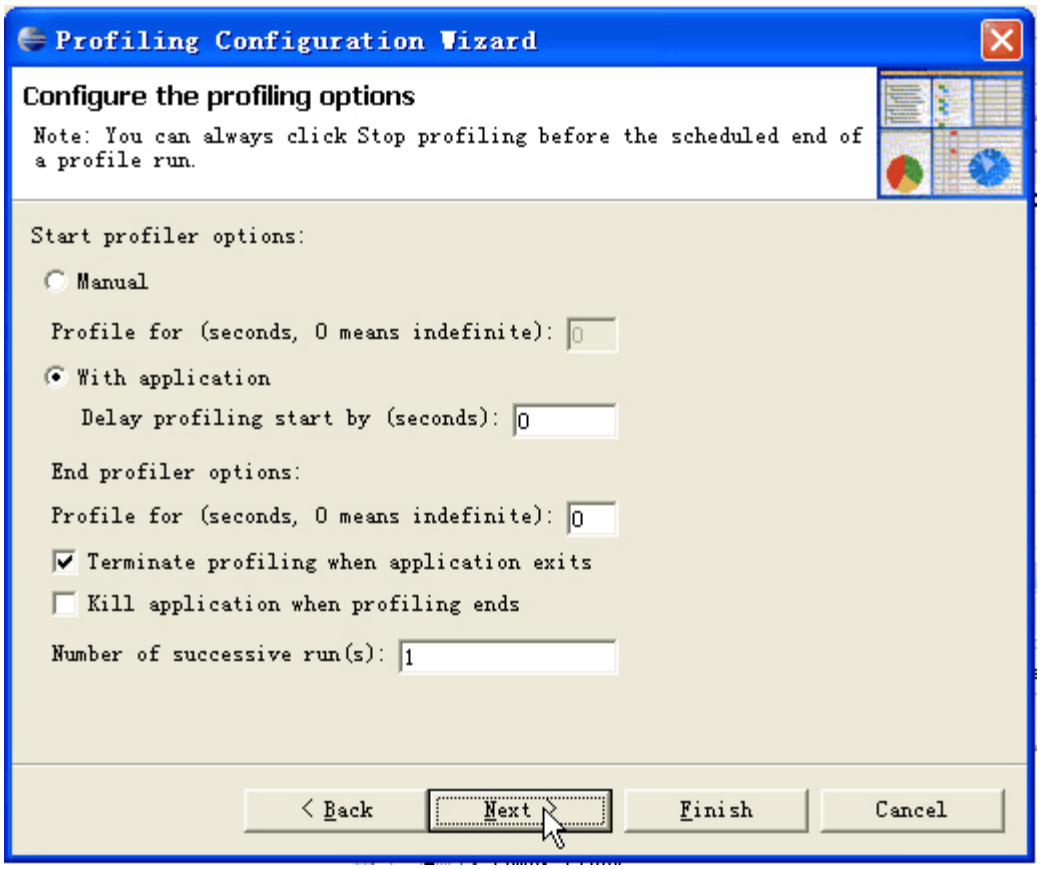
On the second page, choose an application to launch, and enter its command line options and working directory. If the application is a Java application on IBM Virtual Machine for Java, select the Enable Java profiling checkbox, which will define the IBM_JAVA_OPTIONS environment variable for the Java process being started, so that JIT-compiled Java methods are profiled. Note: If you want to profile a system without launching an application, for instance because the application is already running, leave these fields blank. Click Next when this page is complete.



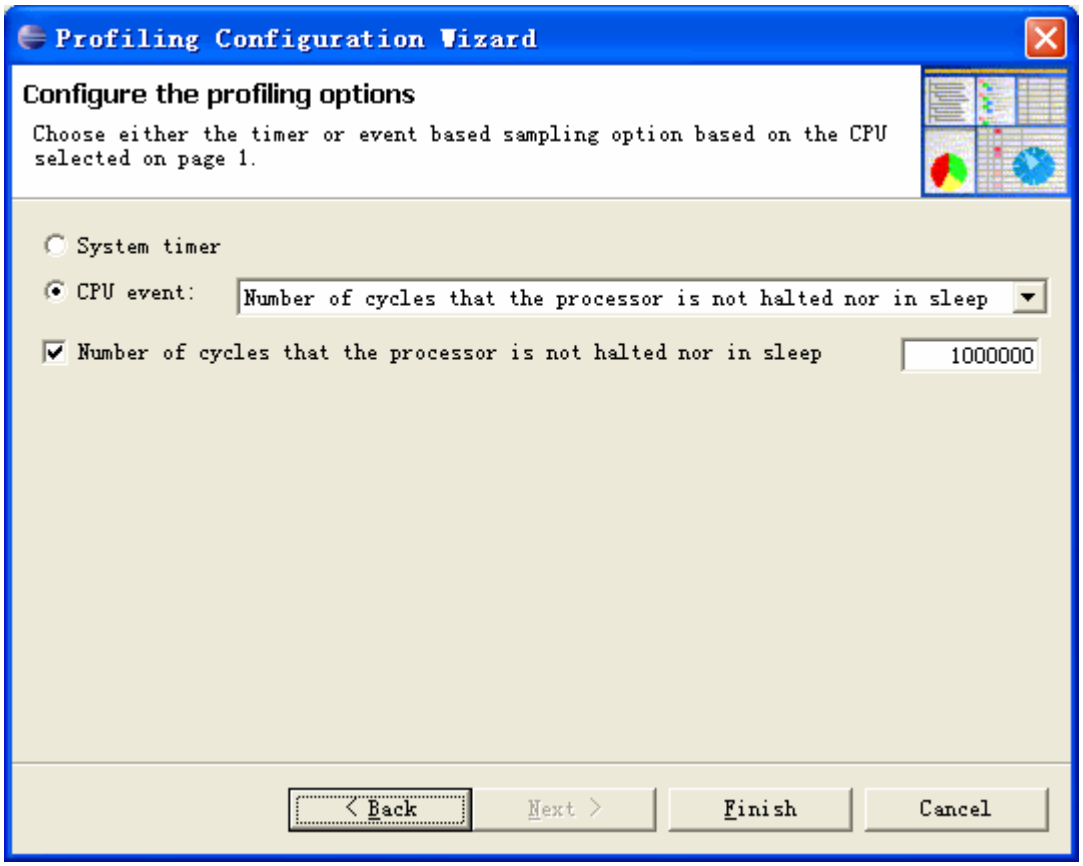
On the third page you can define the way to profile, when and how to start and end the profiling.

Choose whether to profile manually or with the application. (Only manual profiling is available if you did not select an application to launch.) Fully automatic profiling is the simplest: it involves a single click of the Run icon, which will launch the profiler and your application, run the application to completion, stop the profiler, and load the profile into Profile Analyzer. Fully manual profiling requires you to start the profiler, start and stop the application (if one was entered on the second page), and stop the profiler.

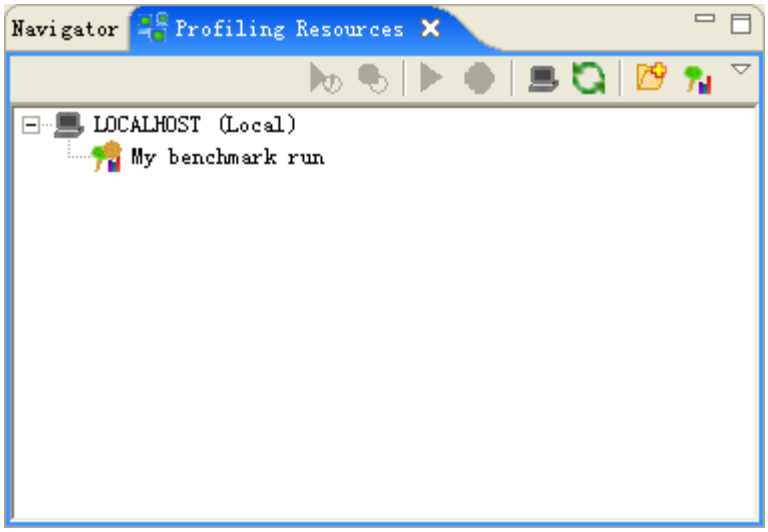
> For fully automatic profiling, choose **With application** and leave the entry fields to their default values of 0. If you want to run your application automatically but give it a predetermined time to "warm up" before profiling begins, choose **With application** and enter the warm-up time in the Delay profiling start by (seconds) entry field.
>
> For fully manual profiling, choose ‘Manual’. You can define the time for profiling in Profile for entry field.



On the final page of the wizard, you can choose a supported CPU event to profile, or leave the default value of System timer. You can also define number of cycles that the processor is not halted or in sleep. Once you have chosen one of these, select **Finish** to create the profiling configuration.



The new profiling configuration should be visible when the host it was created for is expanded:



You can define as many configurations for local systems as you require. If you want to create two configurations that are largely similar but contain slightly different settings (for example, which CPU counter is used or a change in command line arguments), you can make a copy of a configuration as follows:

1. Right-click over the configuration and choose **Make Configuration Copy**. A new configuration is created with the name Copy of original configuration name.
2. Right-click over the copy and choose **Modify Profiling Configuration**.... This starts the Profiling Configuration Wizard. From here you can change any settings in the copied configuration.

5.1.2Run Profiling Configuration

To run a profiling configuration, select the configuration in the Profiling Resources view. You can then use the pop-up menu or the toolbar buttons to start or stop the profiler or the application. Some choices are greyed out from the toolbar, or not shown on the pop-up menu, depending on whether you chose manual or automatic application start.

To start the profiler manually, right-click and choose **Start Profiler**, or click the **Start Profiler** button from the toolbar:  .To start the application manually, or to start both the profiler and the application if you have set the application up to start automatically, right-click and choose **Launch Application**, or click the **Launch Application** button from the toolbar:  .If you have set up automatic profiling, the profile will run according to the configuration settings, and when it ends the profile will be loaded into Profile Analyzer. There is a delay between the end of profiling and the load into Profile Analyzer, which may vary from under a second to several minutes depending on:

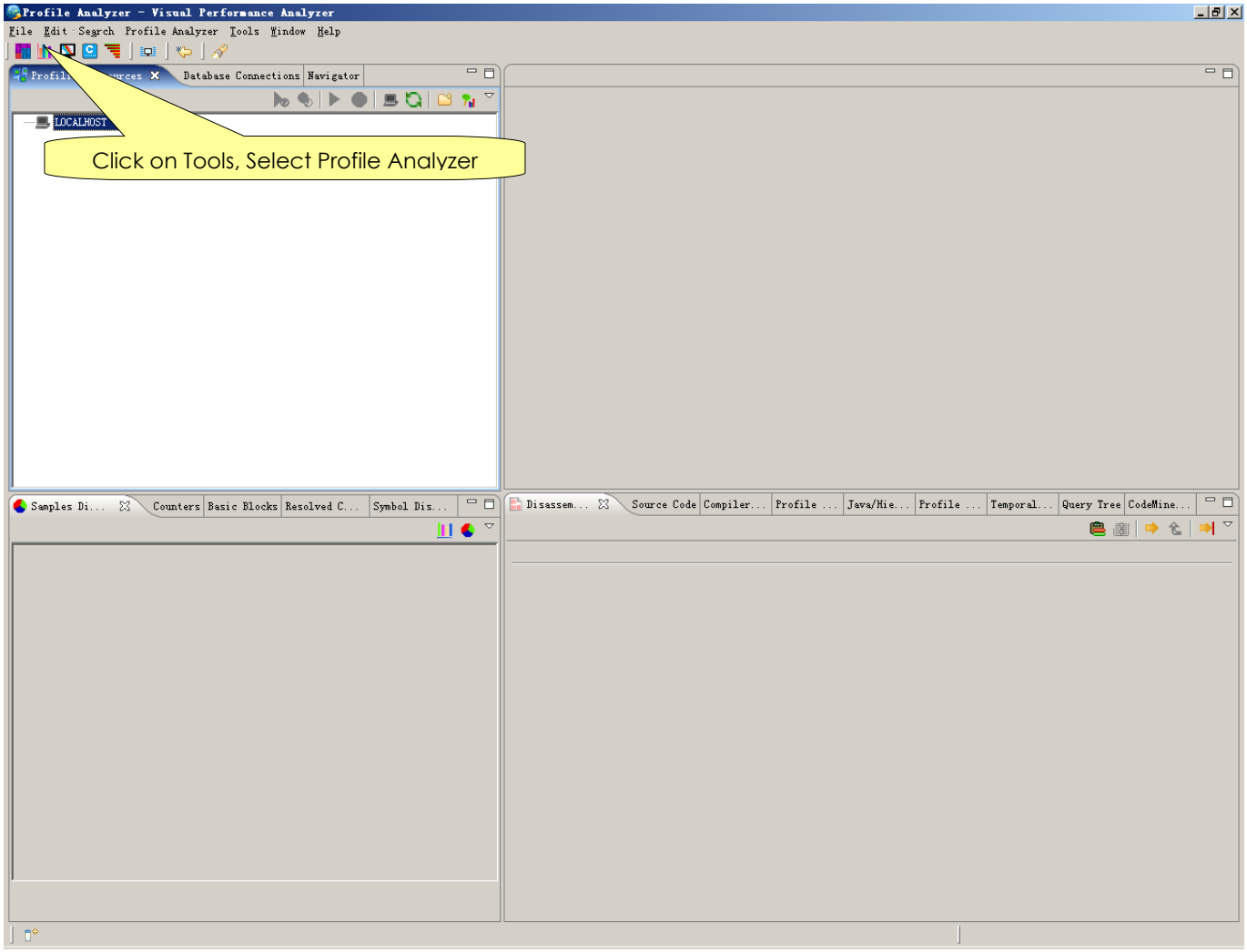
- The amount of profile data (a very long-running profile will have a large trace buffer, which may take a long time to post-process by TPROF before Profile Analyzer can load it)
- For remote profiling, the size of the generated profile and the connection speed.

While the profiling takes place, the Profiling Resources view is greyed out to prevent you from starting multiple profiles at the same time. (This is because the toolbar buttons can only apply to a single running profile.)

You can stop profiling, for a manual configuration or for an automatic configuration where you want to override the automatic settings, by clicking the **Stop Profiler** toolbar button:  .You can stop the profiled application at any time by clicking the **Terminate Application** toolbar button:  .While your application runs, its output is displayed in the Console window (if it produces any output to stdout or stderr). If this window is not visible you can display it by selecting the lower right pane and choosing **Windows -> Show View -> Other -> Profile Analyzer -> Console**.

5.1.3Load an Existing Profile

When you first start Visual Performance Analyzer, press the Tools button and then select Profile Analyzer to load the plug in.



You can also load Profile Analyzer perspective by choosing **Window - Open Perspective - Other - Profile Analyzer**.

If you already have profiles generated by TPROF or a Profile Analyzer compatible XML-based profile generator, you can open them by following the steps below.

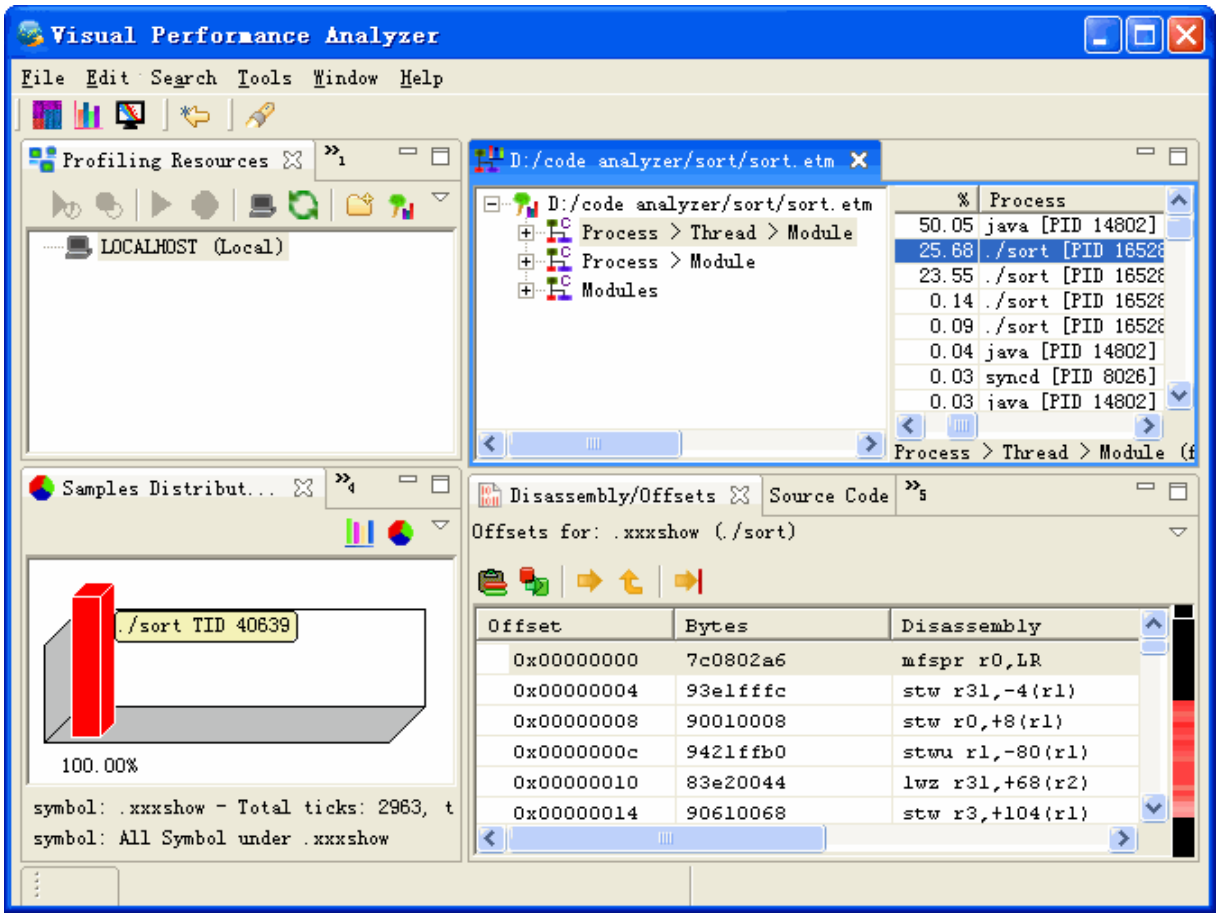
1. Your Eclipse window should look like the one above. Select Profiling Resources view. You can open this view by selecting **Window -> Show View -> Profile Analyzer-> Profiling Resources**.

2. Profile Analyzer profiles must have one of the following extensions:

    - An extension of **.out** , **.etn** or an extension of **.etm**

    - **.opm and .opz**

You can load any of these profiles by click and the profile opens as follows:



In VPA a profile data file loading process is able to run as a background runnable job. When VPA is loading a file, you can click a button to put the loading job to run in the background. While the loading job is running in the background, you can use Profile Analyzer to view already loaded profile data files, or event start another loading job at the same time.

As already stated, profile data files are loaded into database tables and kept in database tables until the user deletes them. Once a profile data file is successfully loaded into a database, further attempts to load the same data file will result in the data being reloaded directly from the database tables. Profile Analyzer does not need to read and parse the original file again. This allows for much faster loading of profile data into VPA after the initial database caching.

Note: although further use of a profile data file results in loading from the database, the original file is still required for Profile Analyzer to work properly. This is because not all of the content of the original file is loaded into database tables. For example, time data is kept in original file and we only store the offset and length information in database tables. When needed, this data is read form they original file on-demand.

5.1.4Profile Navigation

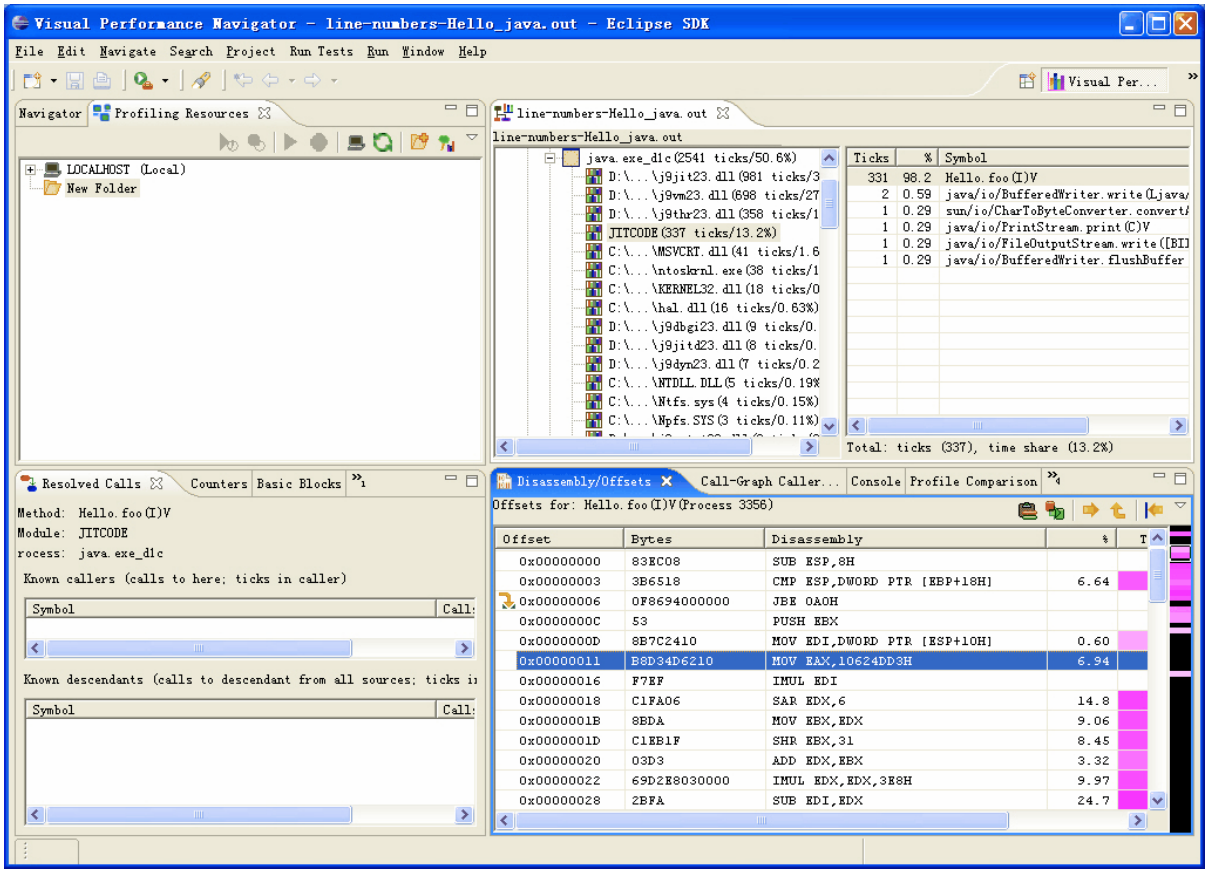
The following are tasks that you can perform to navigate around profiles within Profile Analyzer.

5.1.4.1Navigate process hierarchy

The **Process hierarchy view** appears by default in the top center pane. It shows an expandable list of all processes within the current profile. You can expand a process to view its module, later thread and etc. You can also view the profile in the form of thread or module and etc. Actually, you can define the hierarchy view by right-click profile and choose **Hierarchy Management**. Thread data is not available in merged profiles (.etm extension).

For more information about Hierarchy Management, you can refer to Navigate Generic Hierarchy Model

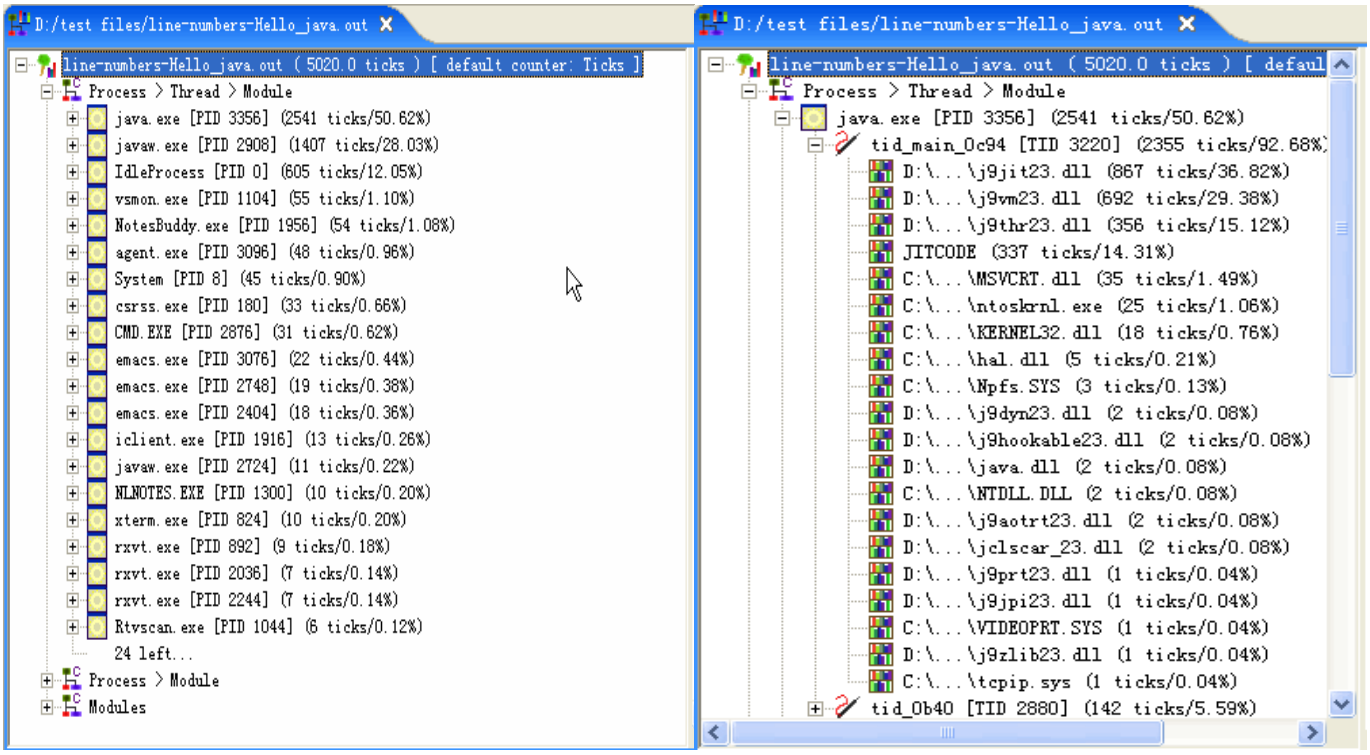
The following screen capture shows a process hierarchy in its unexpanded state:



As in most Profile Analyzer views, objects are sorted from most to fewest ticks. In this view you can see that the **IdleProcess** was the process with the most ticks, indicating either I/O delays or actual idle time during the process (for example, if the application being profiled ran on one CPU and the system had a second, mostly idle CPU).

You can expand a process to view the threads or modules beneath it. As you select a process, thread, or module, the Symbols view updates to display the list of symbols that belong to that process, thread, or module. The Samples Distribution Chart also changes, as you select different processes or threads, to display the proportion of ticks used by the most important modules within the selected process or thread.

The following two views show part of the above process in **Process>Thread>Module** hierarchy:



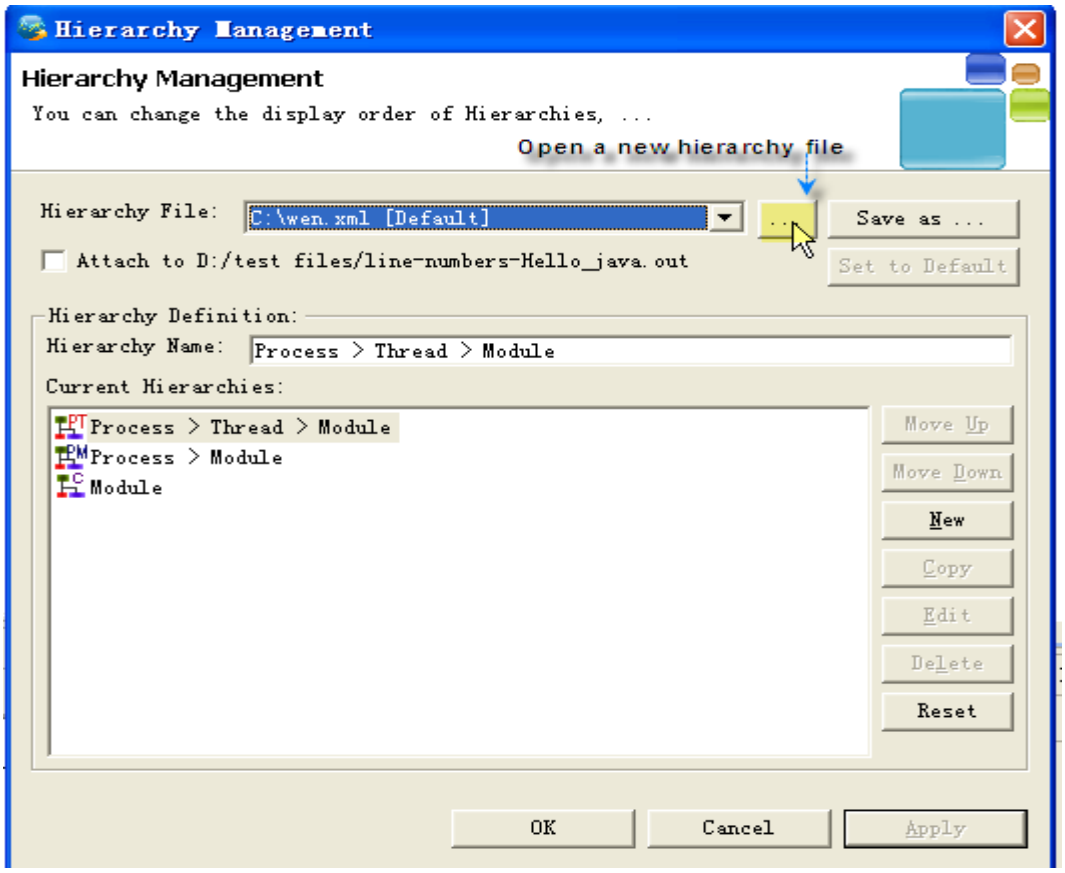
Navigate generic hierarchy model

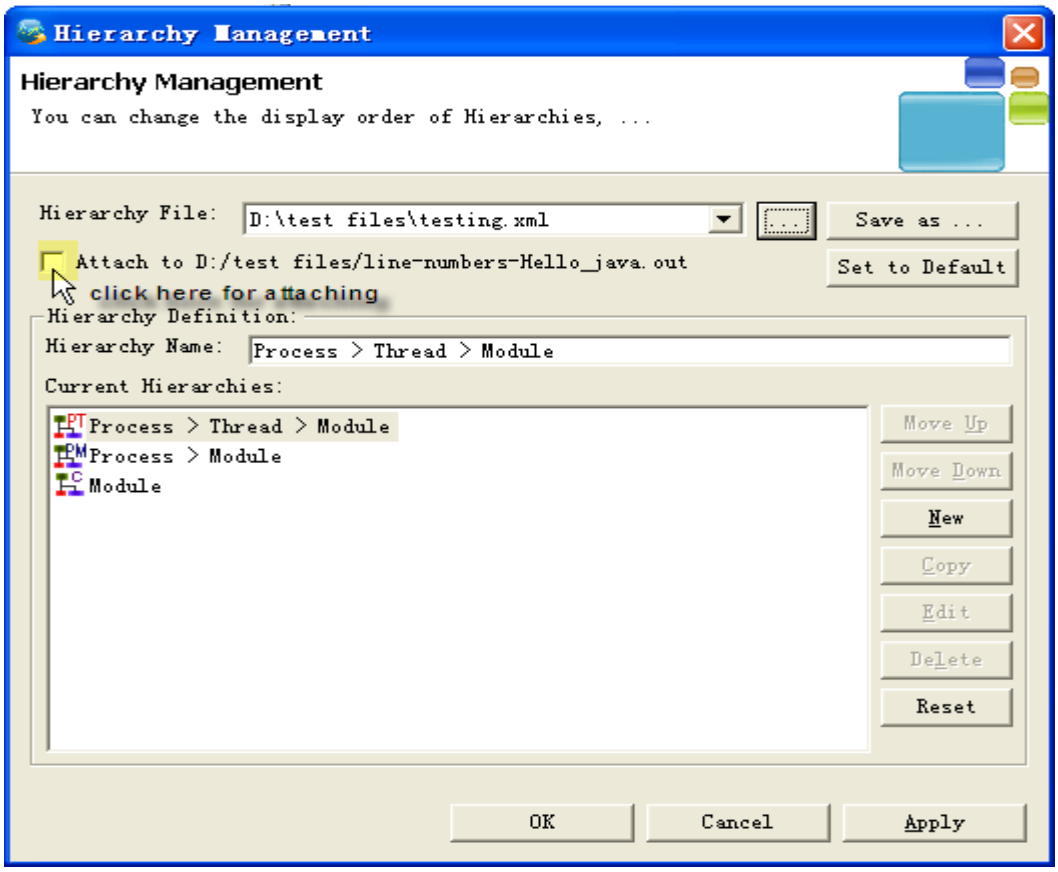
A process may have some threads, and each thread can visit some modules (for instance, DLLs) and call procedures/methods (symbols) in these modules. In default condition, you can observe systems in the hierarchy of **Process > Core > Thread > Module**, **Process > Thread > Module**, **Process-> Module** and **Module**. With the function of generic hierarchy model, you can create your own hierarchy view. For example, if you want to group threads which use a common module, you can display the hierarchy Process > Module > Thread by creating it in the **Hierarchy Management**.

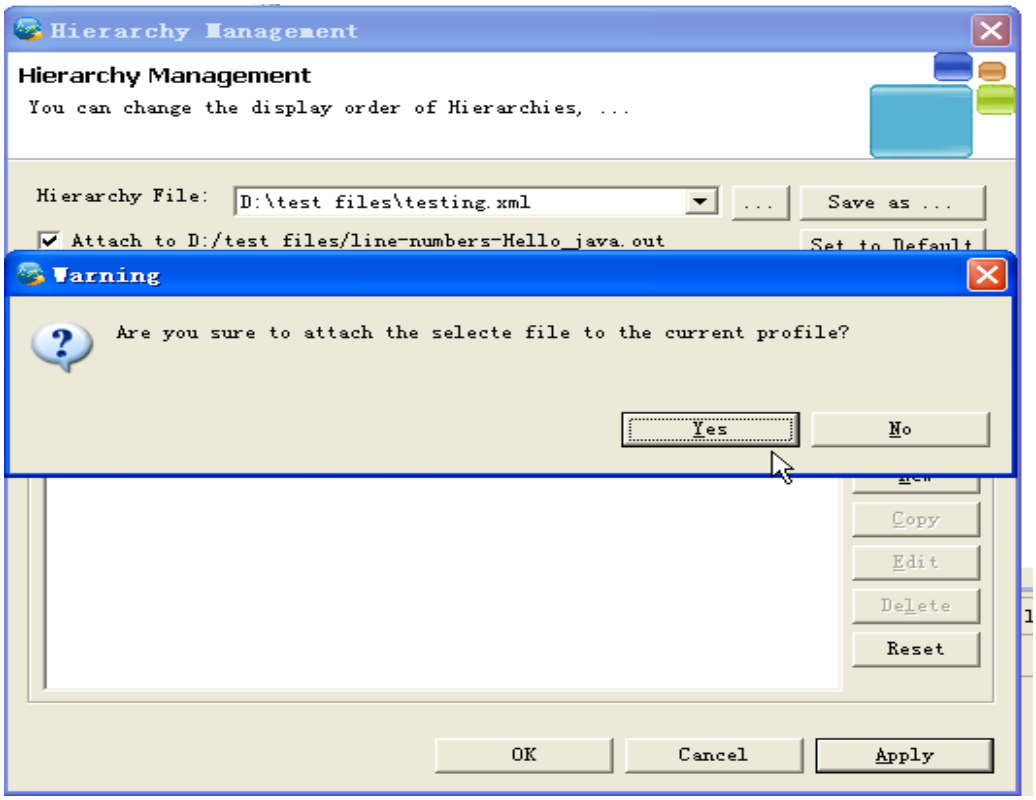
Attach hierarchy file to profile file, please do the following steps:

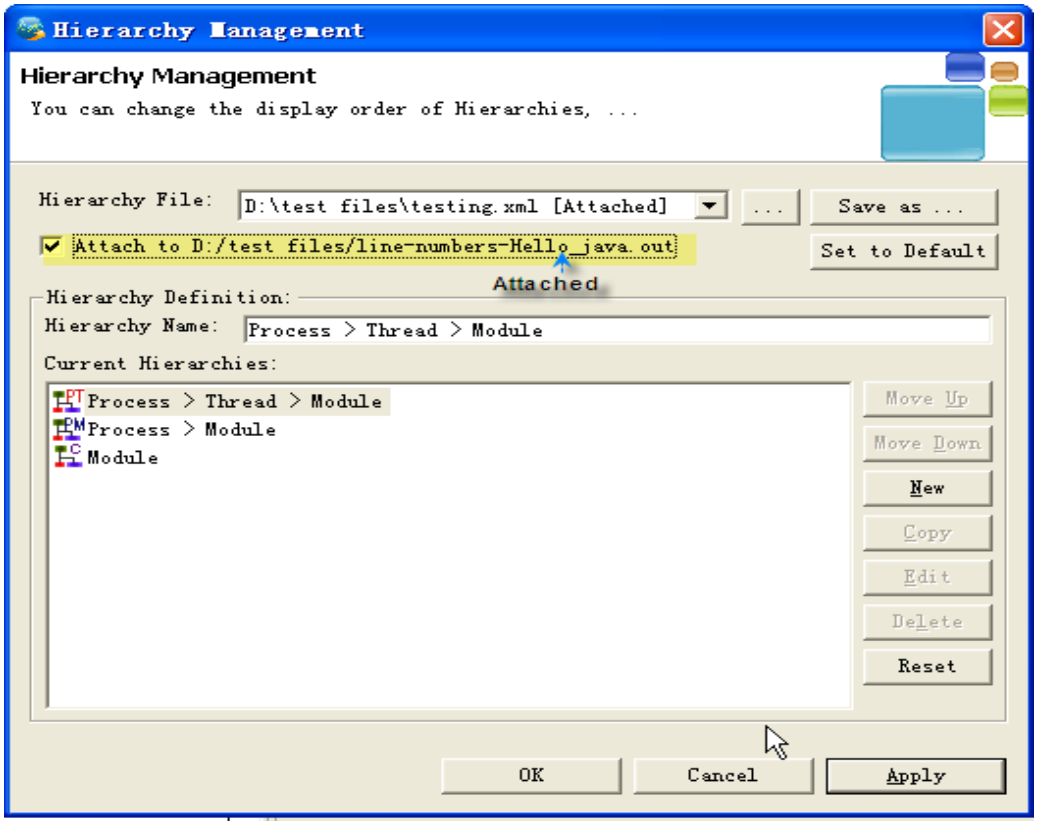
1. Right-click in the process hierarchy view and choose **Hierarchy Management**
2. Click the **...** button to open a new hierarchy file for attaching to the profile file
3. select the check-box to attach the new hierarchy file to the profile file

Here are some pictures to show how to attach a new hierarchy file -“testing.xml”:


Hierarchy Management
Hierarchy Management
You can change the display order of Hierarchies, ...
Open a new hierarchy file
Hierarchy File:
C:\wen.xml [Default]
...
Save as ...
Attach to D:/test files/line-numbers-Hello_java.out
Set to Default
Hierarchy Definition:
Hierarchy Name:
Process > Thread > Module
Current Hierarchies:
Process > Thread > Module
Process > Module
Module
Move Up
Move Down
New
Copy
Edit
Delete
Reset
OK
Cancel
Apply


Hierarchy Management
Hierarchy Management
You can change the display order of Hierarchies, ...
Hierarchy File:
D:\test files\testing.xml
...
Save as ...
Attach to D:/test files/line-numbers-Hello_java.out
Set to Default
click here for attaching
Hierarchy Definition:
Hierarchy Name:
Process > Thread > Module
Current Hierarchies:
Process > Thread > Module
Process > Module
Module
Move Up
Move Down
New
Copy
Edit
Delete
Reset
OK
Cancel
Apply

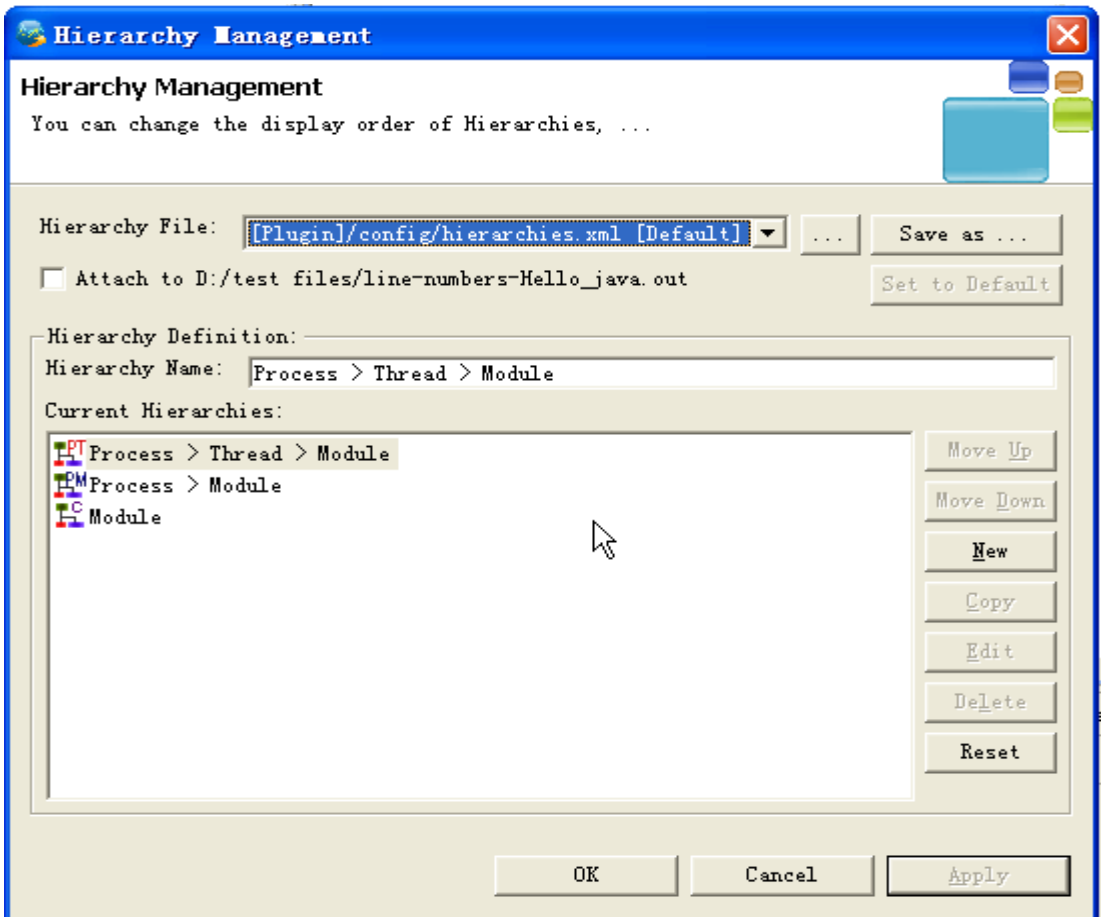

Hierarchy Management
Hierarchy Management
You can change the display order of Hierarchies, ...
Hierarchy File:
D:\test files\testing.xml
...
Save as ...
Attach to D:/test files/line-numbers-Hello_java.out
Set to Default
Warning
Are you sure to attach the selecte file to the current profile?
Yes
No
Copy
Edit
Delete
Reset
OK
Cancel
Apply


Hierarchy Management
Hierarchy Management
You can change the display order of Hierarchies, ...
Hierarchy File:
D:\test files\testing.xml [Attached]
...
Save as ...
Attach to D:/test files/line-numbers-Hello_java.out
Set to Default
Attached
Hierarchy Definition:
Hierarchy Name:
Process > Thread > Module
Current Hierarchies:
Process > Thread > Module
Process > Module
Module
Move Up
Move Down
New
Copy
Edit
Delete
Reset
OK
Cancel
Apply

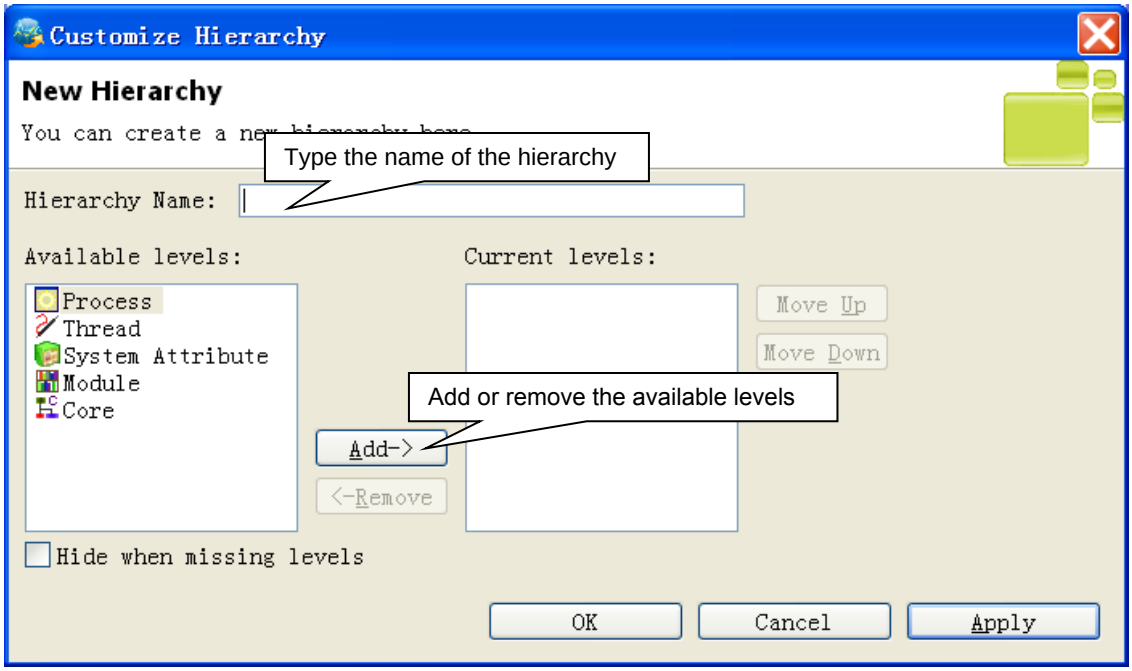
To create you own hierarchy view, follow these steps:

1. Right-click in the process hierarchy view and choose **Hierarchy Management**
2. In the Hierarchy Management Wizard, click **New**
3. Give your hierarchy a specific name if you like, or the system will generate a name for you.
4. Select the element you want to have in your view. You may reorder your hierarchy by choosing **up** or **down**
5. Click **Apply** or **Ok**

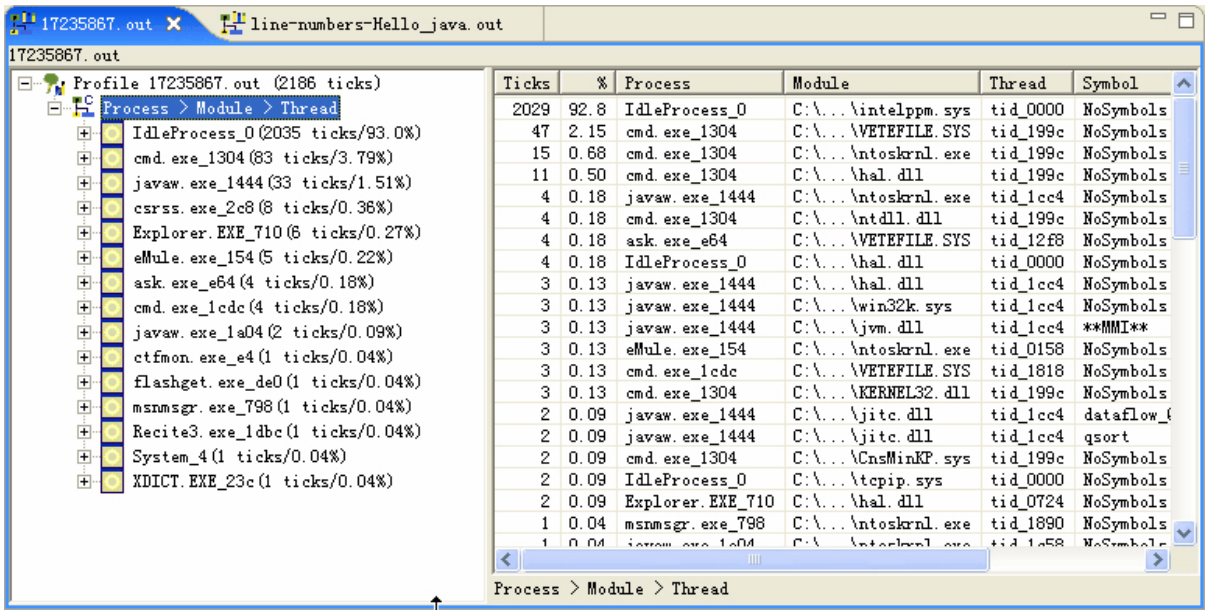
Here is a new hierarchy view we create to see the threads under each module:



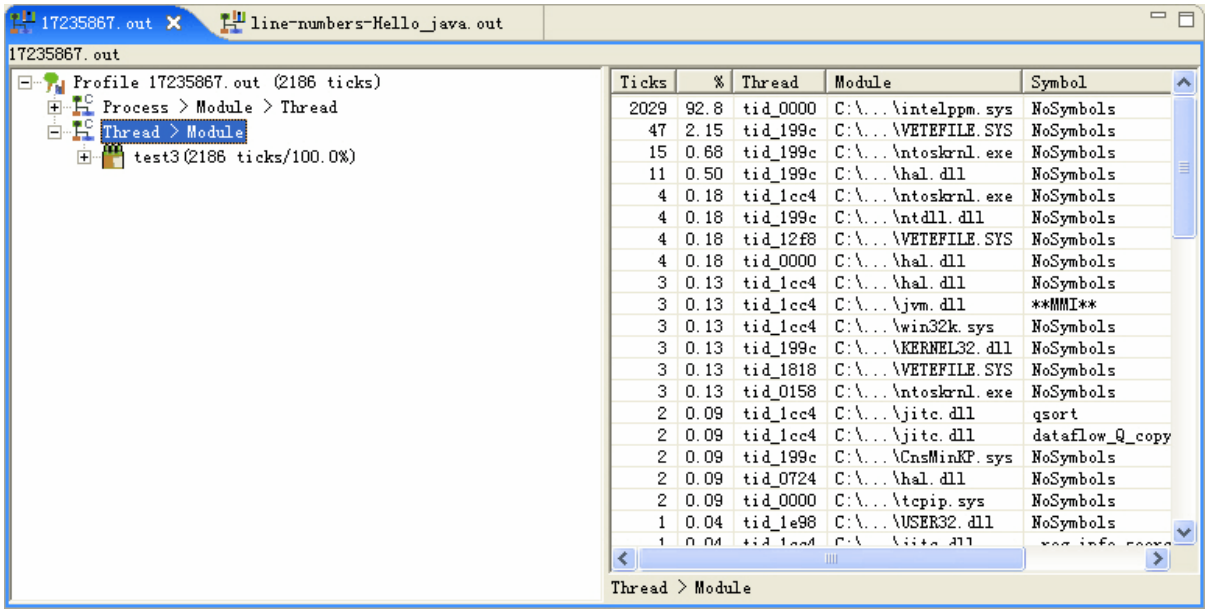
Click the **New** button to create a new hierarchy view:



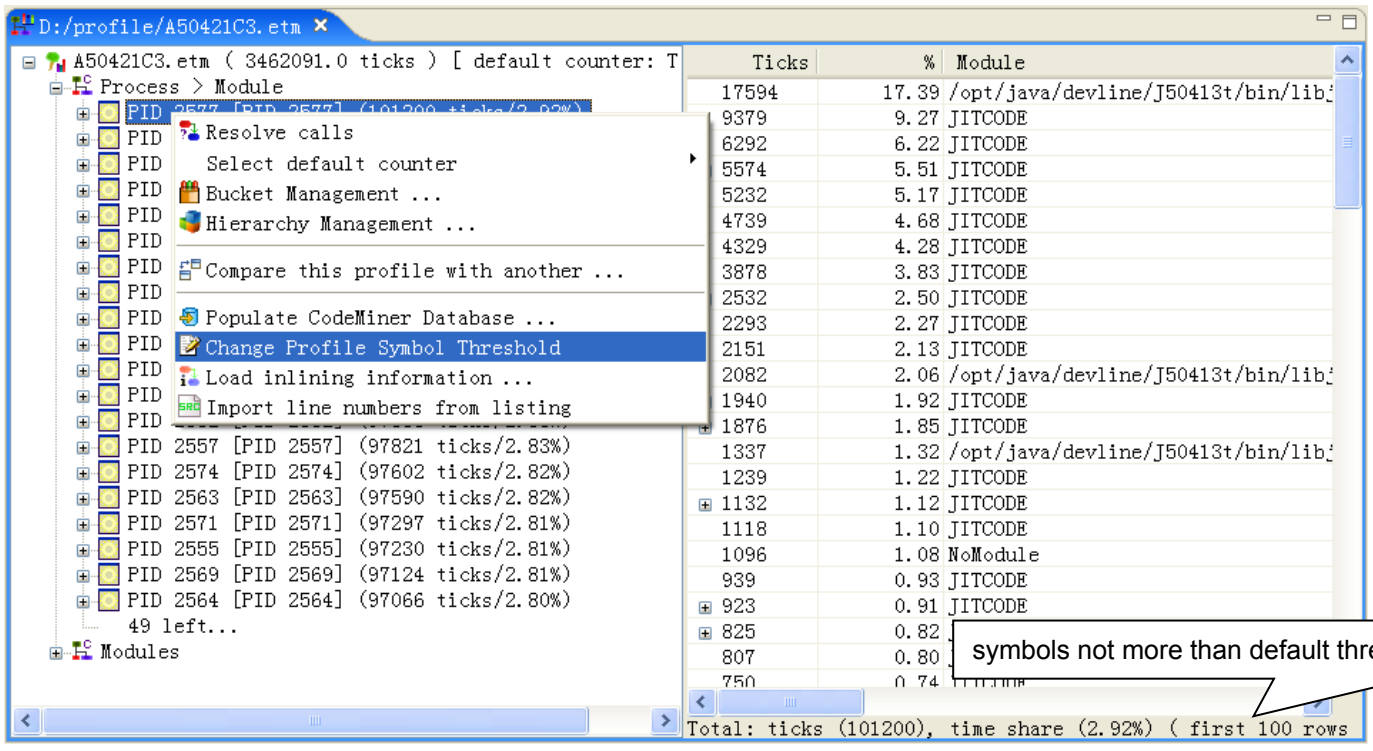
When you click **Apply** or **OK**, you can see the change in **Process Hierarchy View**



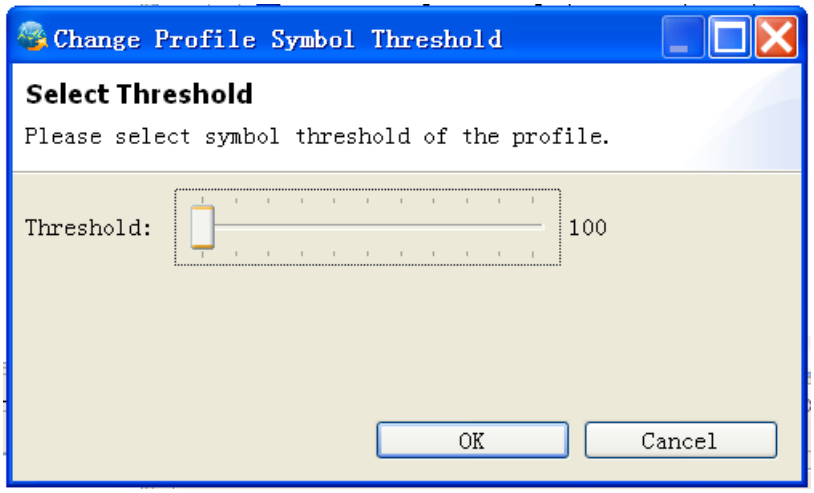
You may add other hierarchy views in the Process Hierarchy View as you like.



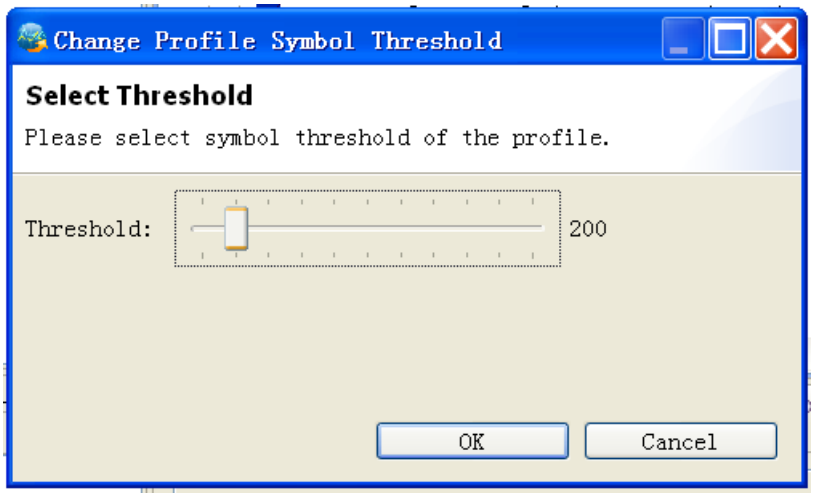
To lines of editor symbol table, user has to set the symbol threshold of the hierarchy. Symbol table contains symbols of the selected hierarchy node, but it does not list all the symbols in default. It often lists no more than a number of them. This is called threshold. After user sets the threshold to another value, the symbol table is refreshed and the listed symbols is no more than the new threshold. The default threshold of editor symbol table is 100, that is, no more than 100 symbols is listed in the table whatever hierarchy node is selected.



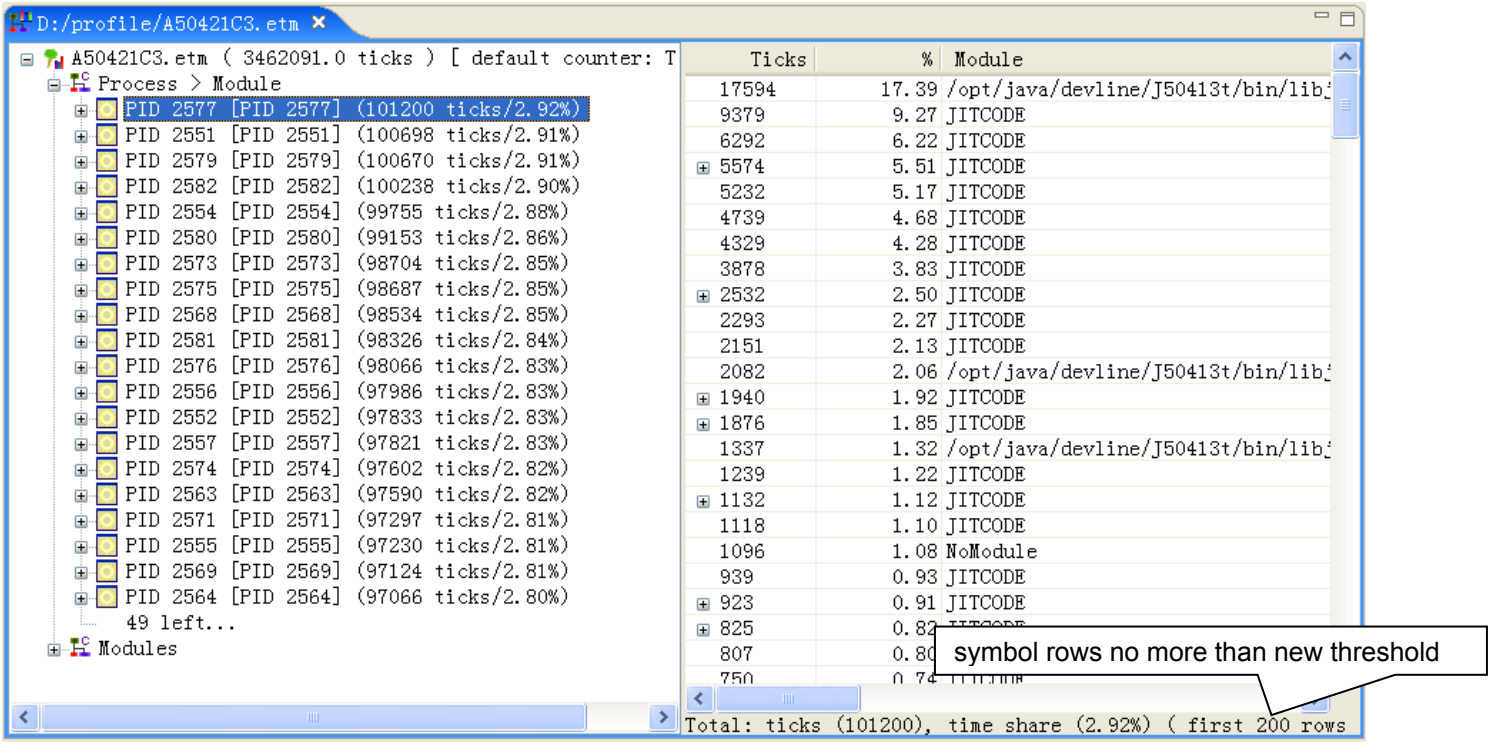
If user selects the **Change Profile Symbol Threshold** item in the context menu, then it pops up the threshold box.



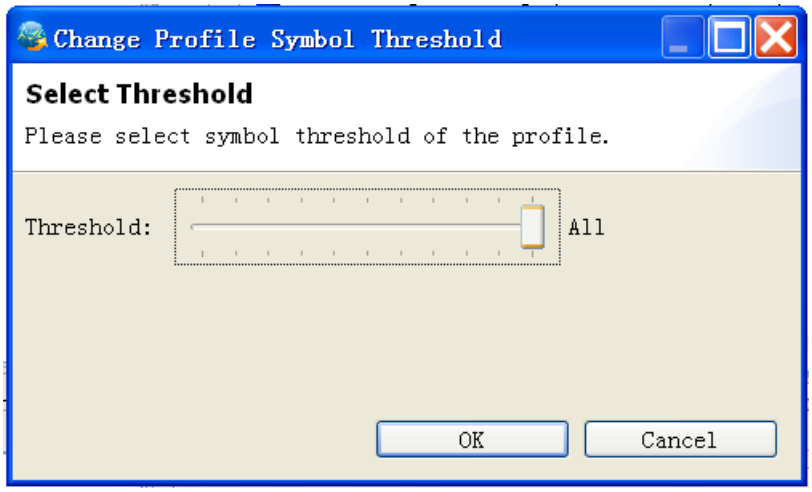
The default symbol threshold is 100. In the symbol table, there are no more than 100 rows listed.



If user sets the threshold to new value 200, the editor symbol table is refreshed and the number of the rows is no more than 200.



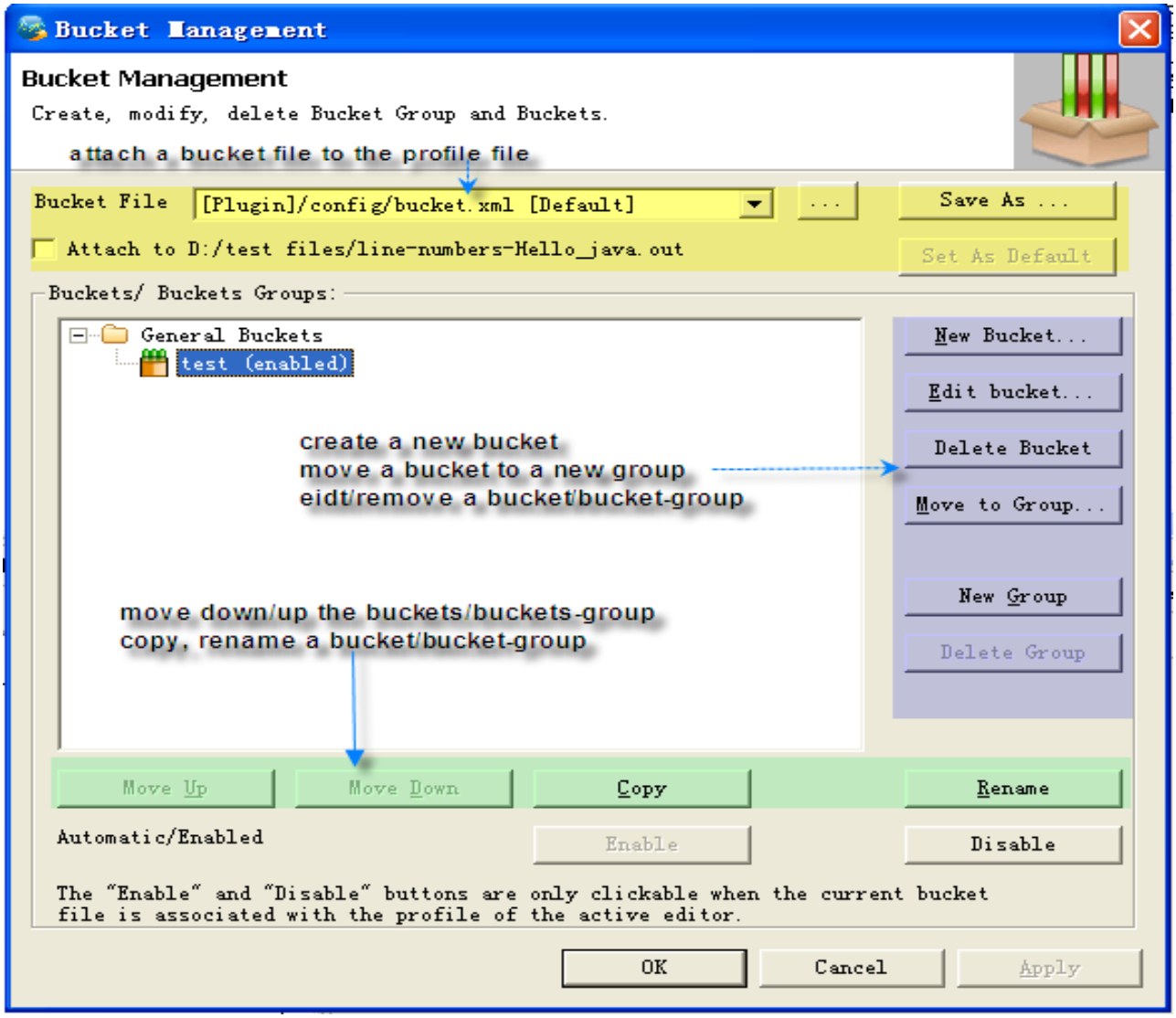
If user set the threshold to ‘All’, symbol table is refreshed and it lists all symbols of the threshold.



5.1.4.2Bucket Management

You can management your buckets settings by doing the followings:

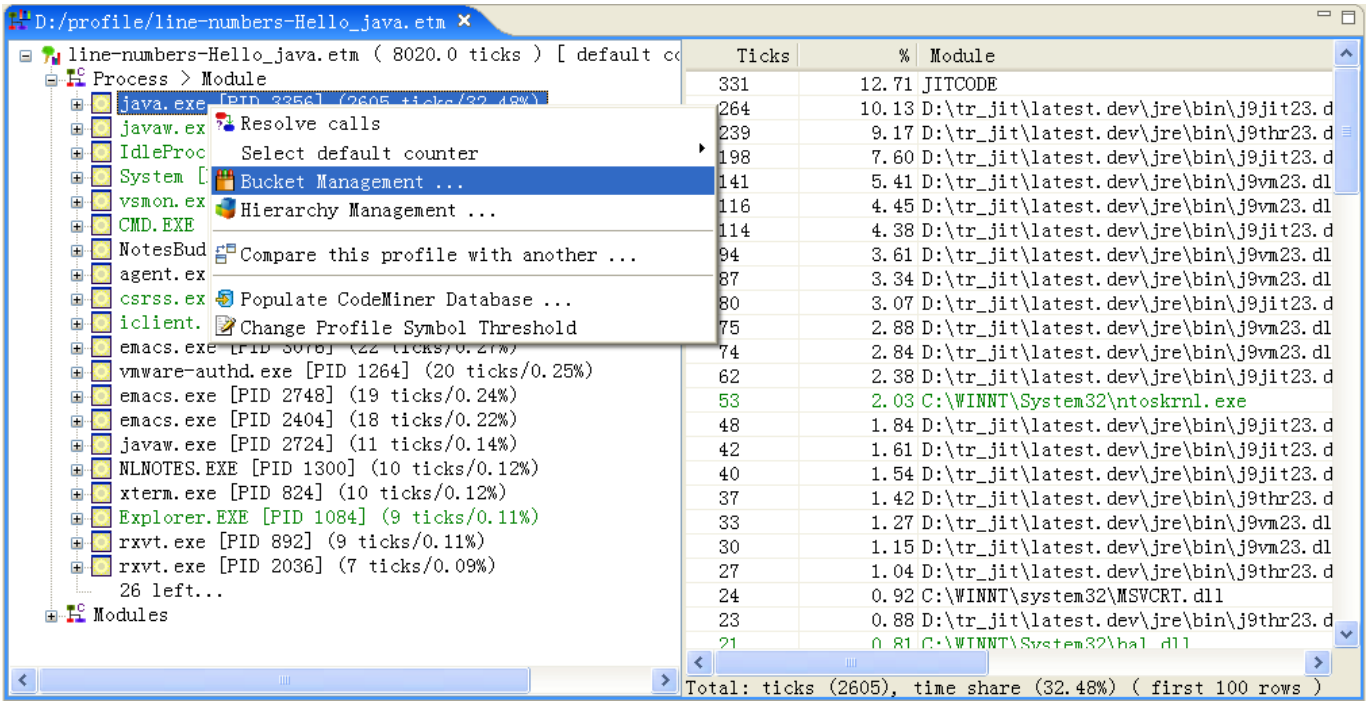
1. Right-click in the process hierarchy view and choose **Buckets-> Bucket Management**
2. Attach a new bucket management file to the profile file
3. Create a new bucket or edit/remove an existing bucket by clicking the corresponding buttons



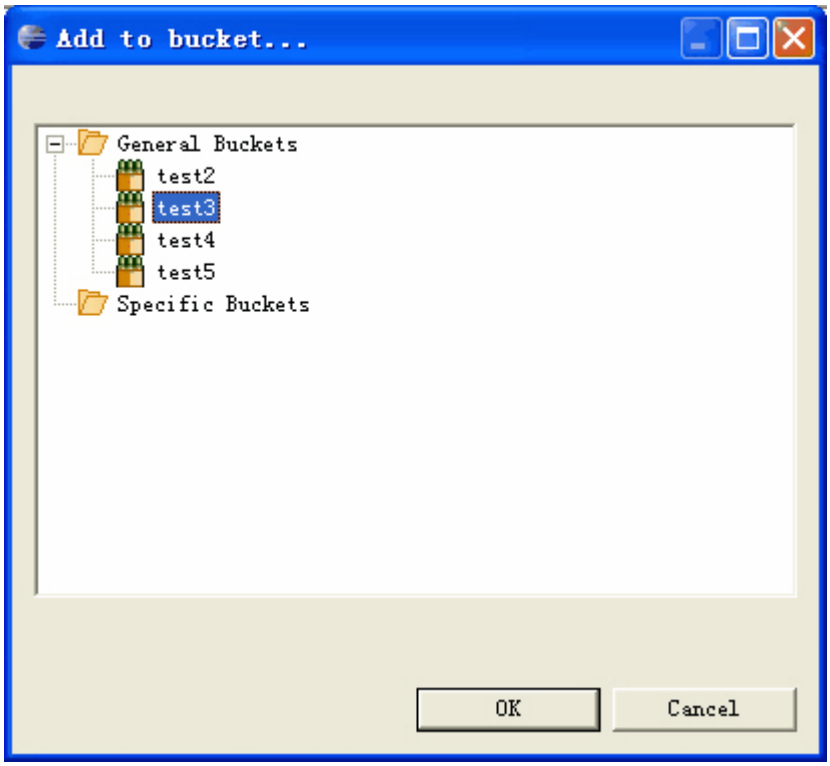
If you have changed bucket definition, the current opened profile will be automatically reloaded. If there are any other opened profiles that are also affected by the new bucket definition, they will not be automatically reloaded. User must reload them manually.

5.1.4.2.1Add new bucket to existing one

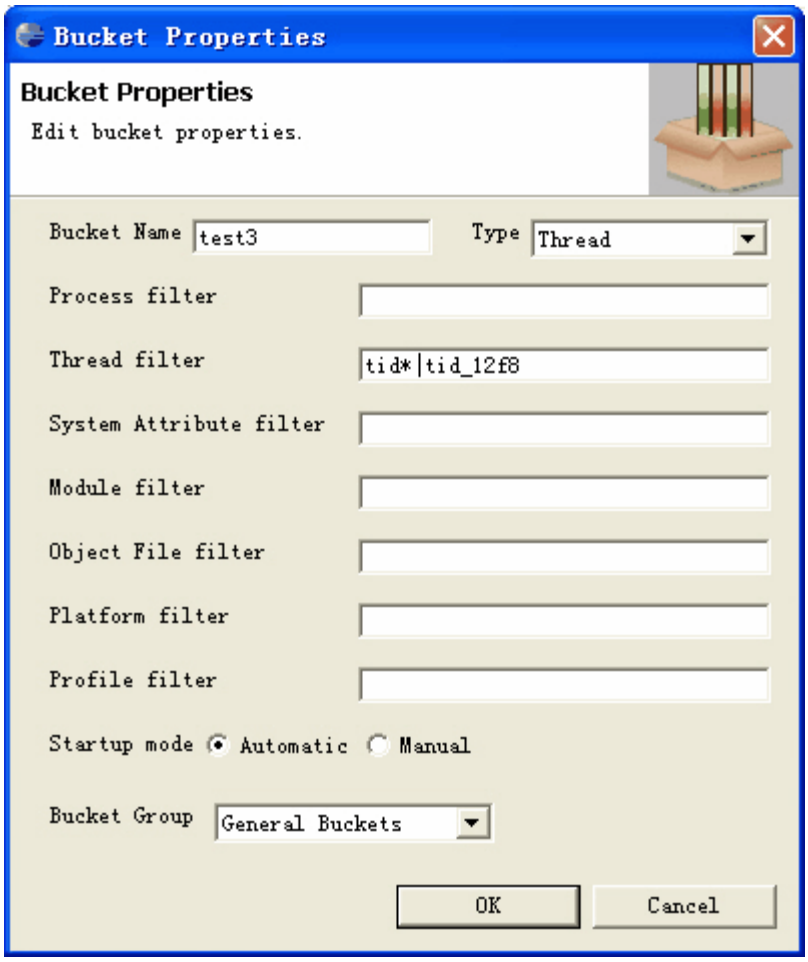
If there are existing buckets to group some components, you can add one to another so as to further filter those components and get the views of those you really want.



If clicked, the bucket selection dialog will open.



After selecting a bucket, the bucket property dialog will open for users to modify the filters as follows:



You can add the filter requirement in this wizard and give this bucket a new name. In order to view this bucket in **Process Hierarchy View**, you may move it up in the **Bucket Management**. You can also edit, delete, enable, disable bucket or create new bucket group in **Bucket Management**.

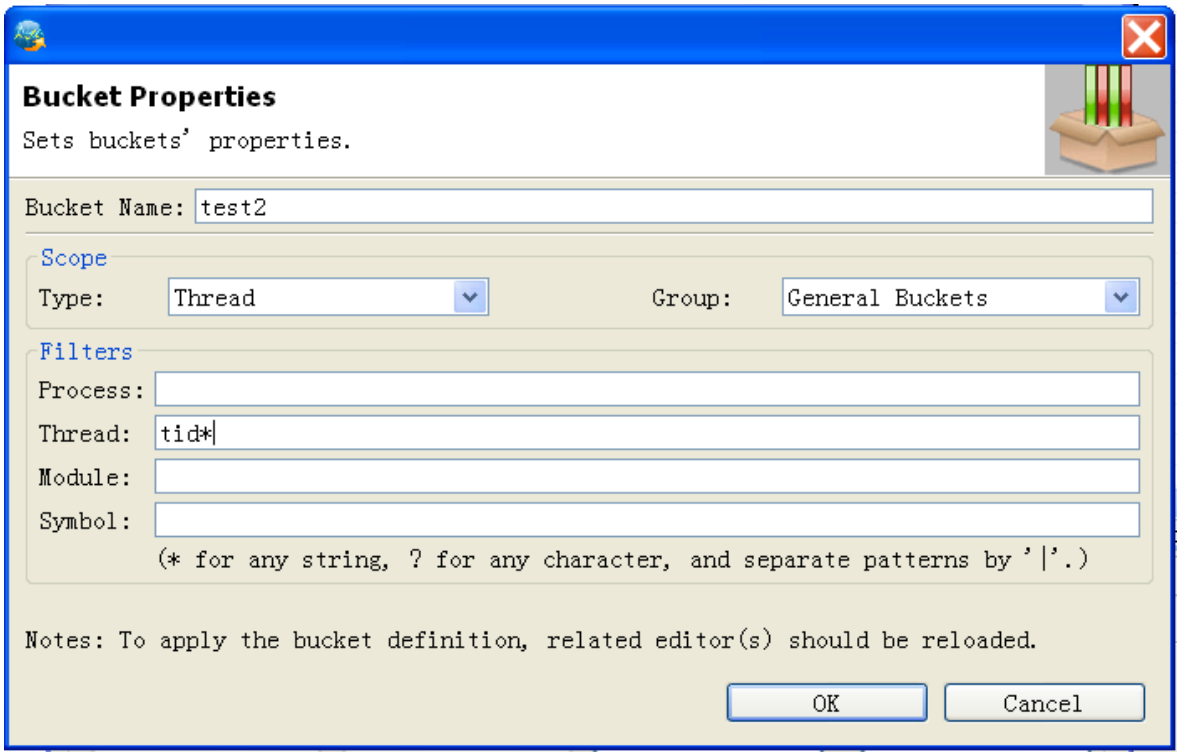
5.1.4.2.2Create new bucket to group common components

In a system, certain groups of modules, threads, or symbols share common components. For example, in a WebSphere java process, threads can be logically grouped by name, with one group containing threads with names like tid_WebContainer*, another with names like tid_Gc_Slave_Thread_*, and another with names like _tid_Alarm_*. Buckets function in Profile Analyzer provides mechanisms for users to group different components, such as objects, together into buckets. It acts as a new layer in the **Hierarchy Process View**.

To create a new bucket, please follow these steps:

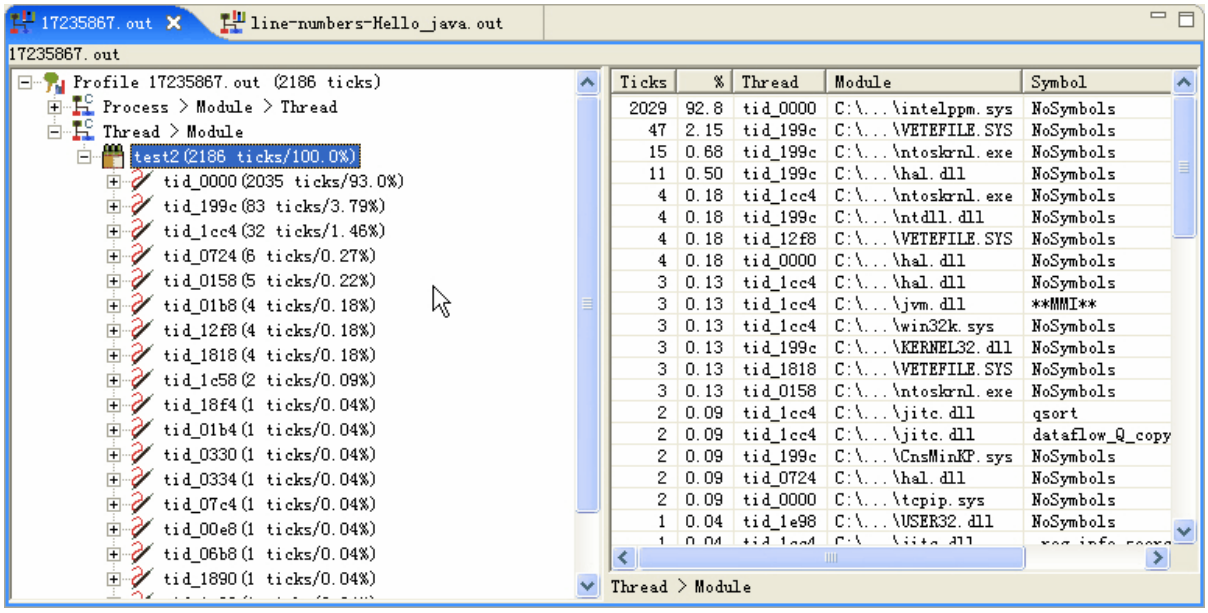
1. Right-click in the process hierarchy view and choose **Buckets-> Create new bucket**
2. Give you **bucket Name** and choose the type of components you want to group from thread, process, module
3. Give the components' name you hope to filter in the corresponding blank, such as tid* thread in thread filter
4. Choose the **bucket group** you want to put your new bucket into.

The following is a example of creating a bucket filtering tid* thread



Now you can define multiple filters for a bucket.

You can see a new layer called test2 appeared above the thread hierarchy view. All the threads with the title beginning **tid** remain as follows:

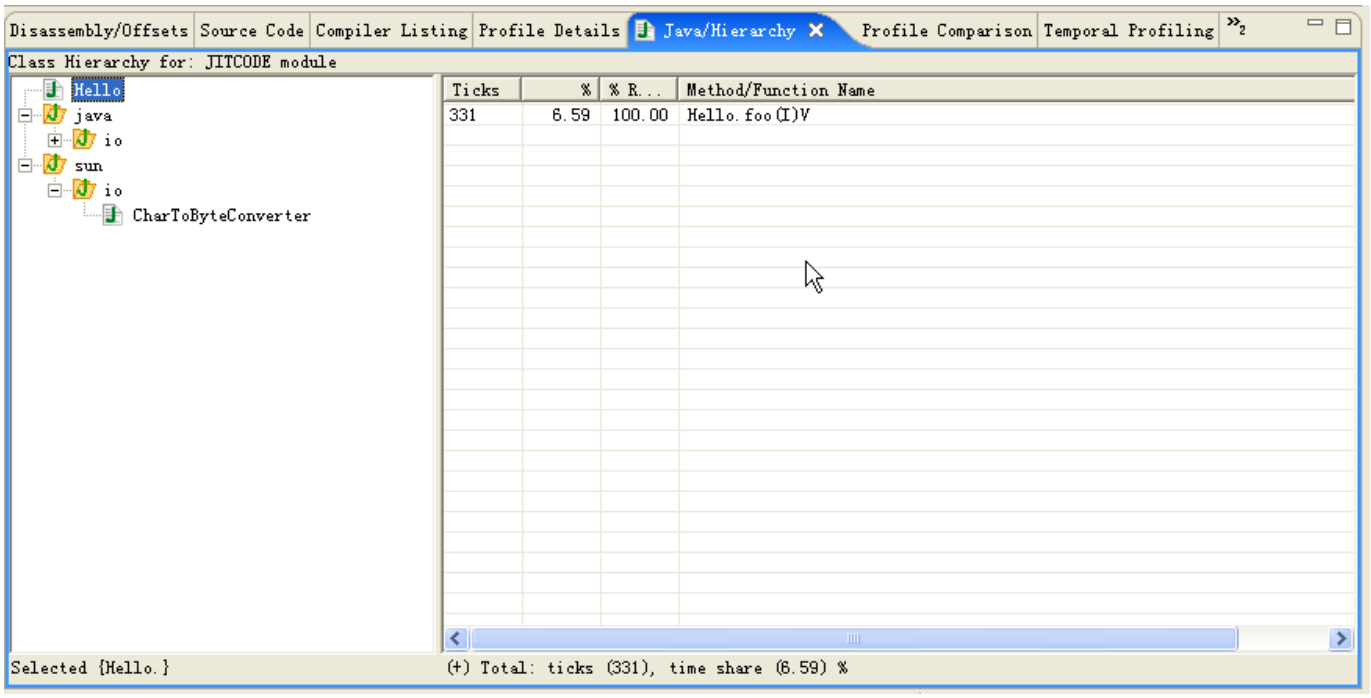


5.1.4.3Navigate Java package hierarchy

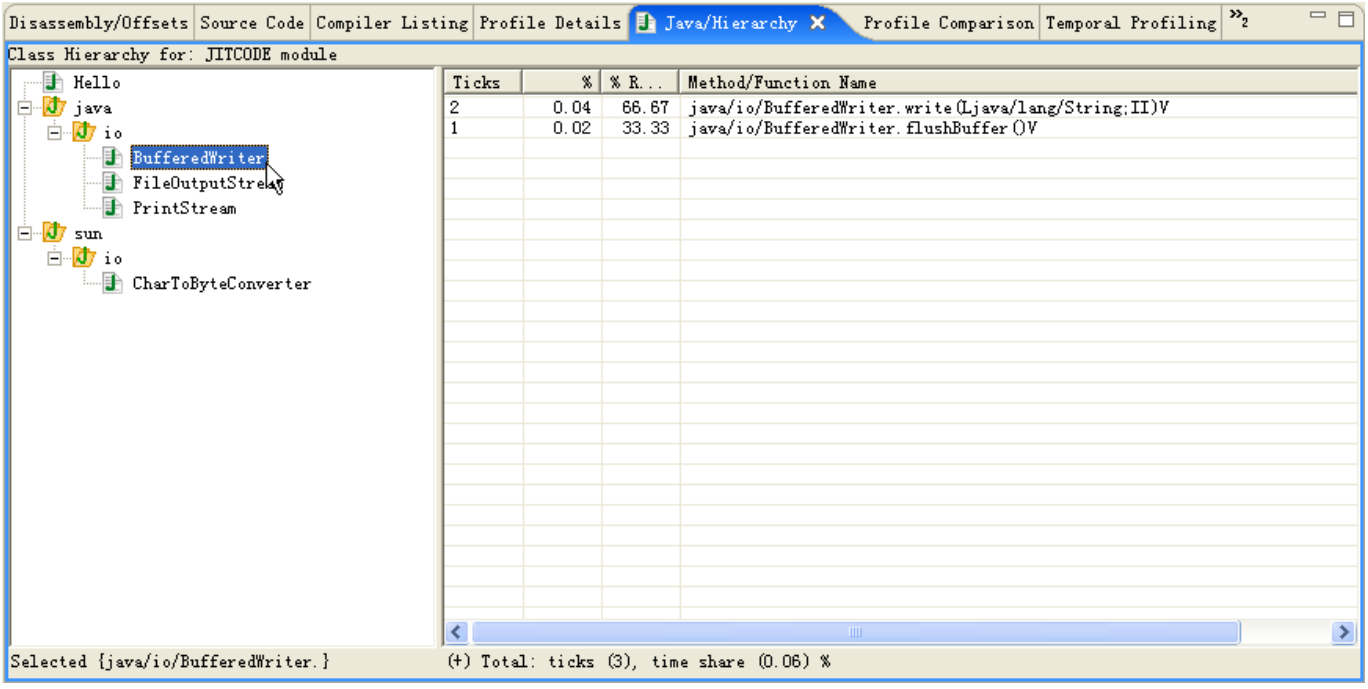
You can view the Java package hierarchy for a process, thread, or module that contains JITted methods using the **Java/Hierarchy** view. The process or thread must contain a JITCODE module, and the module must be the JITCODE module.

Note: The Java/Classes hierarchy view is normally displayed at the bottom right, along with the **Disassembly/Offsets** view, the **Temporal Profiling** view, and the Profiling Configurations **Console** view. If you cannot see it displayed, you can open it with **Windows -> Show View -> Other -> Profile Analyzer-> Java/Classes hierarchy**.

To view the Java package hierarchy for a process, thread, or JITCODE module, click on the **Java/Classes hierarchy** tab, then navigate the **Process hierarchy** view. As you select different processes, threads, or modules, the Java/classes hierarchy is updated to show you the Java package hierarchy for any active methods. The following screen shot shows an initial view of the Java package hierarchy for a JITCODE module on a AIX profile:



You can select a top-level name to view all methods in all packages that match that name (for example java). Or you can expand a top-level name to display the packages and classes beneath it. By selecting a package or class in the hiearchy, you can limit the list of displayed methods to those in that class or hierarchy. The following shows the active methods for the BufferedWriter class in the java.io package:



When you double-click on a method in the table at the right, the following views are updated to display information for that method:

- The OffsetAsm Information view
- The Disassembly Resovled Call Information view

5.1.4.3.1Notes on appropriate use of this view

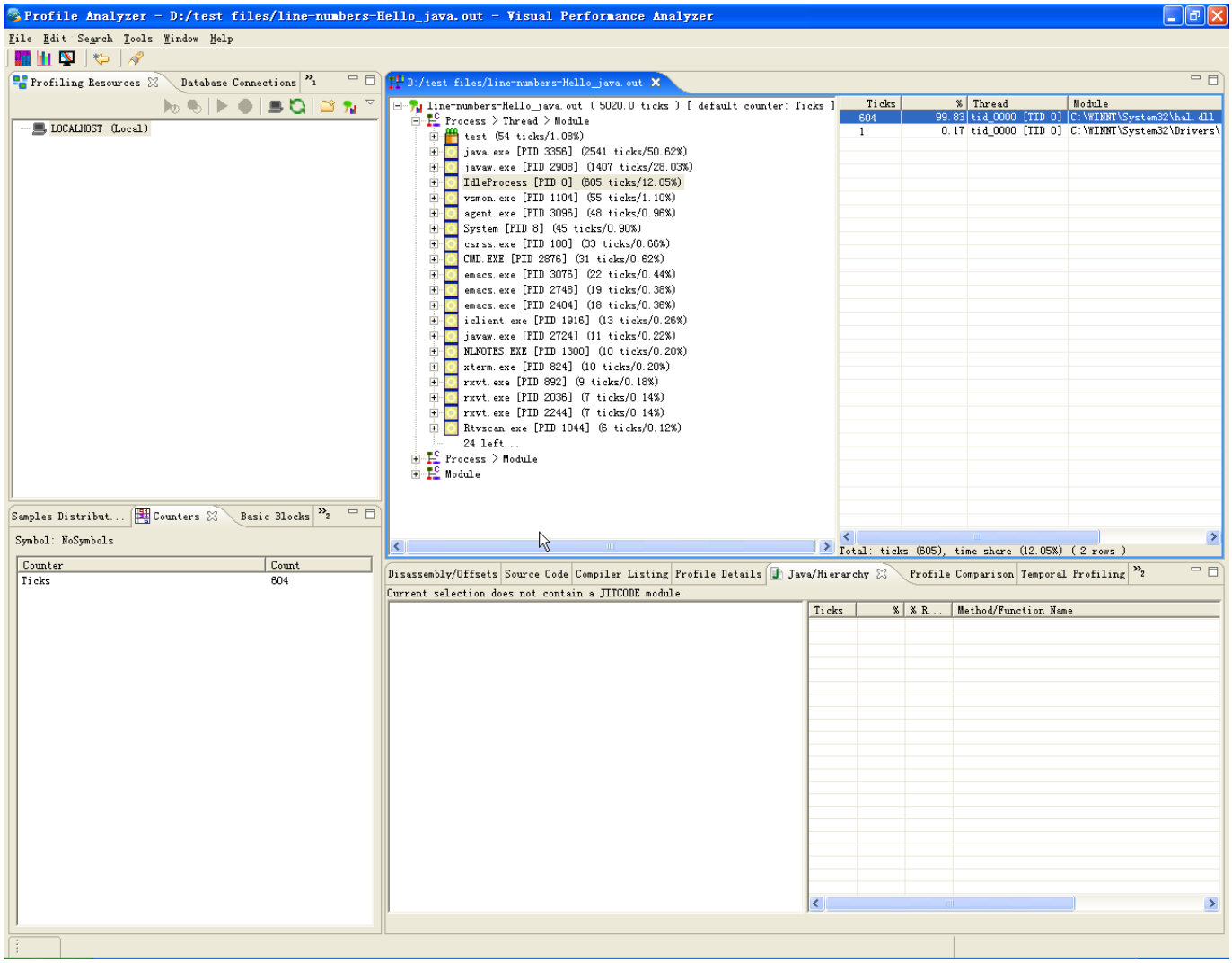
The Java/Classes Hierarchy view is useful if you are working on tuning the code for specific classses under your control and are trying to determine what bottlenecks exist in those classes. However, you should avoid the pitfall of focusing simply on the classes you have control over (classes that you can make source code changes to). For example, trying to tune the hottest method in a class you own may provide some benefit, but if that method takes only 1% of total ticks, while java/io/BufferedWrite.write takes 20%, you should look at what methods are calling java/io/BufferedWrite.write (using the Disassembly Resovled Call Information view on Windows, Linux-IA32, and Linux-x86-64, or using a call graph profiling tool such as ITRACE on other platforms).

Conversely, just because the methods in your packages do not show significant CPU usage does not mean they are efficiently written or have no impact on performance. For instance, if a significant percentage of profile time is spent in the JVM garbage collection library (e.g. libj9gc23.so or j9gc23.dll), this may indicate that you are making inefficient use of memory by allocating too many objects or failing to make them available to garbage collection when they are no longer needed.

5.1.4.4View counters

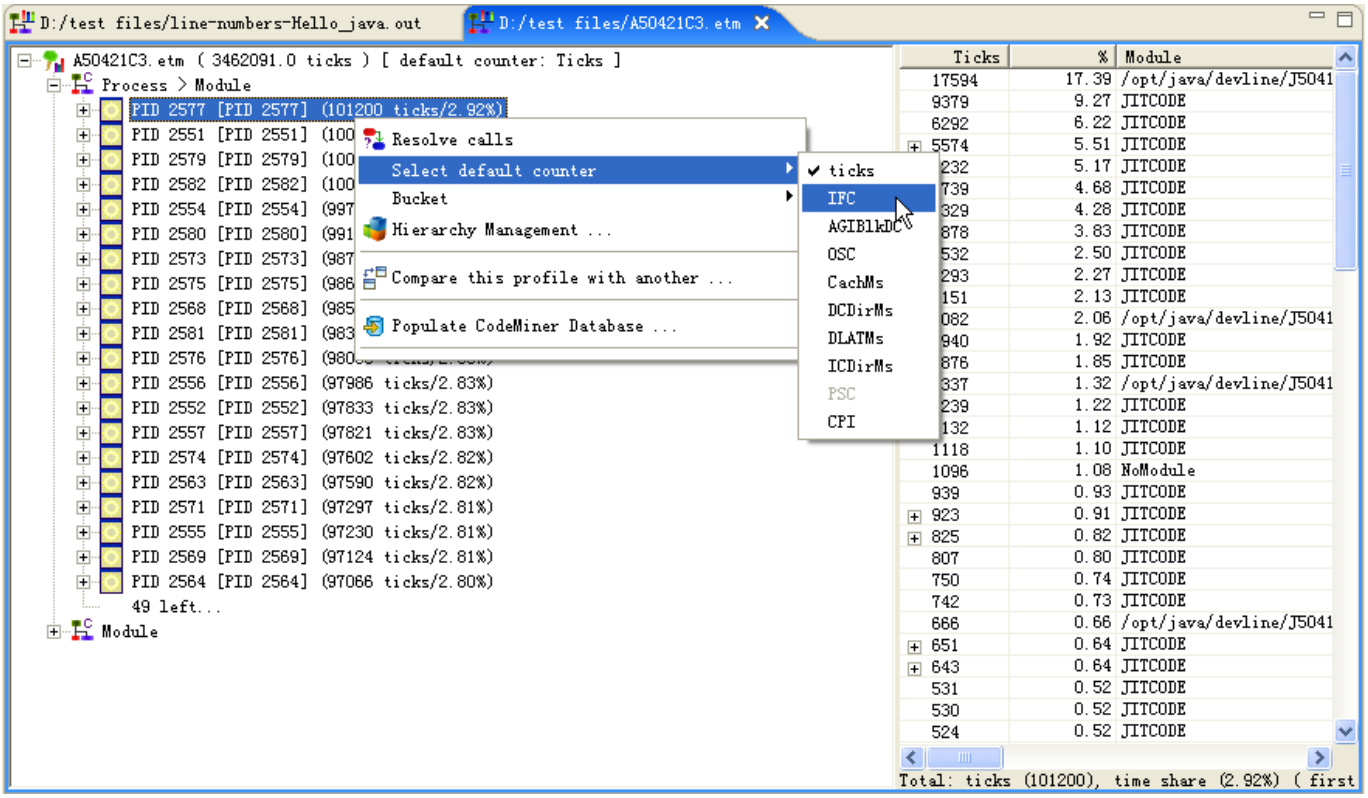
In **Counters** view, you can view the ticks of process, thread, module or bucket you selected in the **Process Hierarchy View**. To open this view, choose **Window -> Show View -> Others->Profile Analyzer -> Counters** or just find it in left pane.

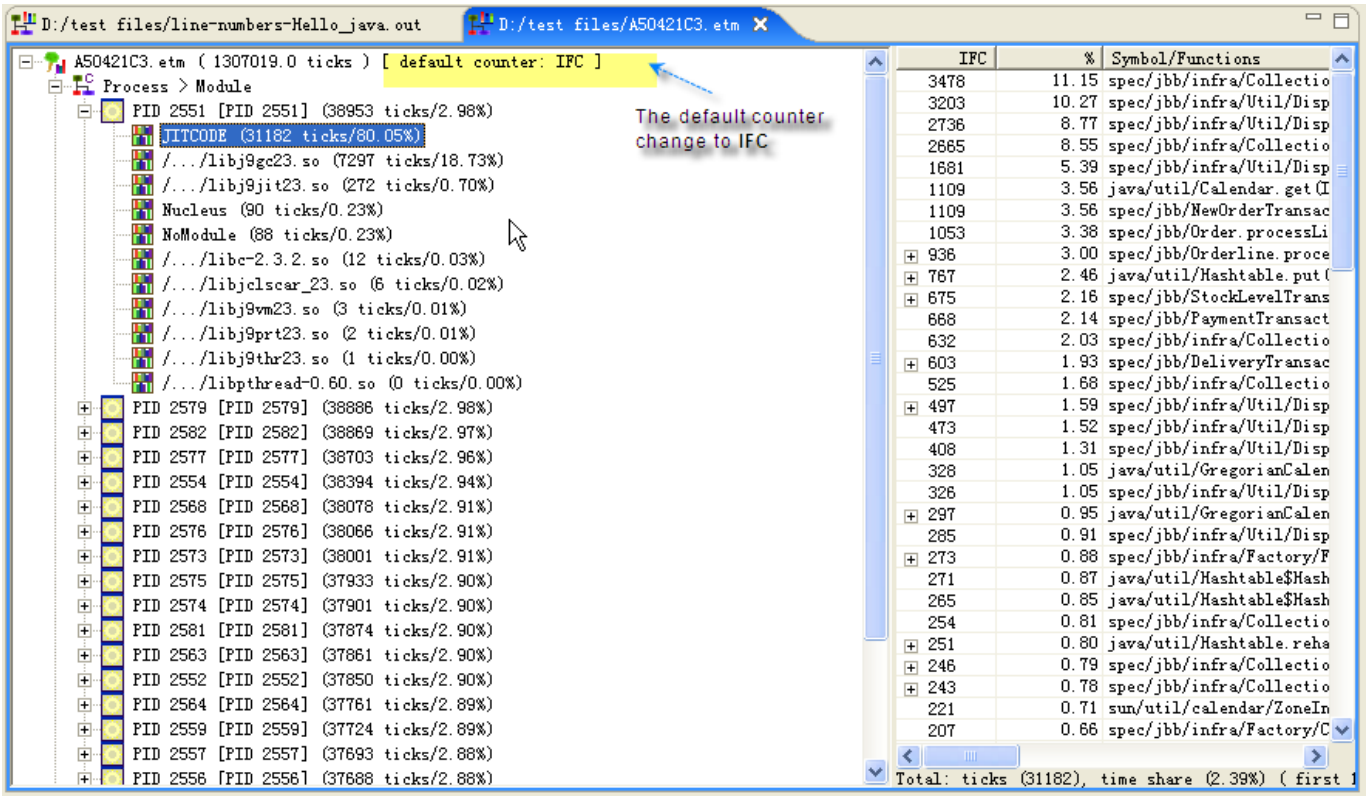
For example, if you select a process in the **Process Hierarchy View**, you can see the total ticks of this process in the **Counters** view



5.1.4.5Select default counter

Whenever you have a profile that contains more than one counter, Profile Analyzer will allow you to choose which is the default counter, used for sorting. Profile Analyzer supports the “sort by counter” feature. A user can select any available counter as the default “sort” counter. Once the default counter is selected, all editors and views understand this selection and will sort/format their outputs according to the current active “sort” counter. Following picture shows this feature:


D:/test files/line-numbers-Hello_java.out
D:/test files/A50421C3.etm
A50421C3.etm (3462091.0 ticks) [default counter: Ticks]
Process > Module
PID 2577 [PID 2577] (101200 ticks/2.92%)
PID 2551 [PID 2551] (100
PID 2579 [PID 2579] (100
PID 2582 [PID 2582] (100
PID 2554 [PID 2554] (997
PID 2580 [PID 2580] (991
PID 2573 [PID 2573] (987
PID 2575 [PID 2575] (986
PID 2568 [PID 2568] (985
PID 2581 [PID 2581] (983
PID 2576 [PID 2576] (980
PID 2556 [PID 2556] (97986 ticks/2.83%)
PID 2552 [PID 2552] (97833 ticks/2.83%)
PID 2557 [PID 2557] (97821 ticks/2.83%)
PID 2574 [PID 2574] (97602 ticks/2.82%)
PID 2563 [PID 2563] (97590 ticks/2.82%)
PID 2571 [PID 2571] (97297 ticks/2.81%)
PID 2555 [PID 2555] (97230 ticks/2.81%)
PID 2569 [PID 2569] (97124 ticks/2.81%)
PID 2564 [PID 2564] (97066 ticks/2.80%)
49 left...
Module
Resolve calls
Select default counter
Bucket
Hierarchy Management ...
Compare this profile with another ...
Populate CodeMiner Database ...
ticks
IFC
AGIBlkDC
OSC
CachMs
DCDirMs
DLATMs
ICDirMs
PSC
CPI
Ticks
%
Module
17594 17.39 /opt/java/devline/J5041
9379 9.27 JITCODE
6292 6.22 JITCODE
5574 5.51 JITCODE
232 5.17 JITCODE
739 4.68 JITCODE
329 4.28 JITCODE
878 3.83 JITCODE
532 2.50 JITCODE
293 2.27 JITCODE
151 2.13 JITCODE
082 2.06 /opt/java/devline/J5041
940 1.92 JITCODE
876 1.85 JITCODE
337 1.32 /opt/java/devline/J5041
239 1.22 JITCODE
132 1.12 JITCODE
1118 1.10 JITCODE
1096 1.08 NoModule
939 0.93 JITCODE
923 0.91 JITCODE
825 0.82 JITCODE
807 0.80 JITCODE
750 0.74 JITCODE
742 0.73 JITCODE
666 0.66 /opt/java/devline/J5041
651 0.64 JITCODE
643 0.64 JITCODE
531 0.52 JITCODE
530 0.52 JITCODE
524 0.52 JITCODE
Total: ticks (101200), time share (2.92%) (first

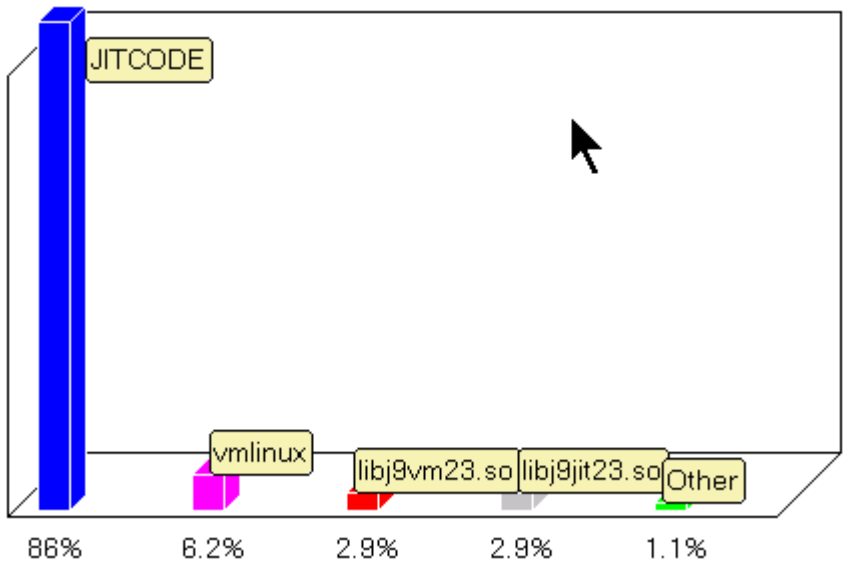

D:/test files/line-numbers-Hello_java.out
D:/test files/A50421C3.etm
A50421C3.etm (1307019.0 ticks) [default counter: IFC]
Process > Module
PID 2551 [PID 2551] (38953 ticks/2.98%)
JITCODE (31182 ticks/80.05%)
/.../libj9gc23.so (7297 ticks/18.73%)
/.../libj9jit23.so (272 ticks/0.70%)
Nucleus (90 ticks/0.23%)
NoModule (88 ticks/0.23%)
/.../libc-2.3.2.so (12 ticks/0.03%)
/.../libjclscar_23.so (6 ticks/0.02%)
/.../libj9vm23.so (3 ticks/0.01%)
/.../libj9prt23.so (2 ticks/0.01%)
/.../libj9thr23.so (1 ticks/0.00%)
/.../libpthread-0.60.so (0 ticks/0.00%)
PID 2579 [PID 2579] (38886 ticks/2.98%)
PID 2582 [PID 2582] (38869 ticks/2.97%)
PID 2577 [PID 2577] (38703 ticks/2.96%)
PID 2554 [PID 2554] (38394 ticks/2.94%)
PID 2568 [PID 2568] (38078 ticks/2.91%)
PID 2576 [PID 2576] (38066 ticks/2.91%)
PID 2573 [PID 2573] (38001 ticks/2.91%)
PID 2575 [PID 2575] (37933 ticks/2.90%)
PID 2574 [PID 2574] (37901 ticks/2.90%)
PID 2581 [PID 2581] (37874 ticks/2.90%)
PID 2563 [PID 2563] (37861 ticks/2.90%)
PID 2552 [PID 2552] (37850 ticks/2.90%)
PID 2564 [PID 2564] (37761 ticks/2.89%)
PID 2559 [PID 2559] (37724 ticks/2.89%)
PID 2557 [PID 2557] (37693 ticks/2.88%)
PID 2556 [PID 2556] (37688 ticks/2.88%)
The default counter change to IFC
IFC
%
Symbol/Functions
3478 11.15 spec/jbb/infra/Collectio
3203 10.27 spec/jbb/infra/Util/Disp
2736 8.77 spec/jbb/infra/Util/Disp
2665 8.55 spec/jbb/infra/Collectio
1681 5.39 spec/jbb/infra/Util/Disp
1109 3.56 java/util/Calendar.get(I
1109 3.56 spec/jbb/NewOrderTransac
1053 3.38 spec/jbb/Order.processLi
936 3.00 spec/jbb/Orderline.proce
767 2.46 java/util/Hashtable.put(
675 2.16 spec/jbb/StockLevelTrans
668 2.14 spec/jbb/PaymentTransact
632 2.03 spec/jbb/infra/Collectio
603 1.93 spec/jbb/DeliveryTransac
525 1.68 spec/jbb/infra/Collectio
497 1.59 spec/jbb/infra/Util/Disp
473 1.52 spec/jbb/infra/Util/Disp
408 1.31 spec/jbb/infra/Util/Disp
328 1.05 java/util/GregorianCalen
326 1.05 spec/jbb/infra/Util/Disp
297 0.95 java/util/GregorianCalen
285 0.91 spec/jbb/infra/Util/Disp
273 0.88 spec/jbb/infra/Factory/F
271 0.87 java/util/Hashtable$Hash
265 0.85 java/util/Hashtable$Hash
254 0.81 spec/jbb/infra/Collectio
251 0.80 java/util/Hashtable.reha
246 0.79 spec/jbb/infra/Collectio
243 0.78 spec/jbb/infra/Collectio
221 0.71 sun/util/calendar/ZoneIn
207 0.66 spec/jbb/infra/Factory/C
Total: ticks (31182), time share (2.39%) (first 1

5.1.4.6View module sample distribution

The **Samples distribution chart** shows a tick distribution for modules in the currently selected process or thread in the process hierarchy.

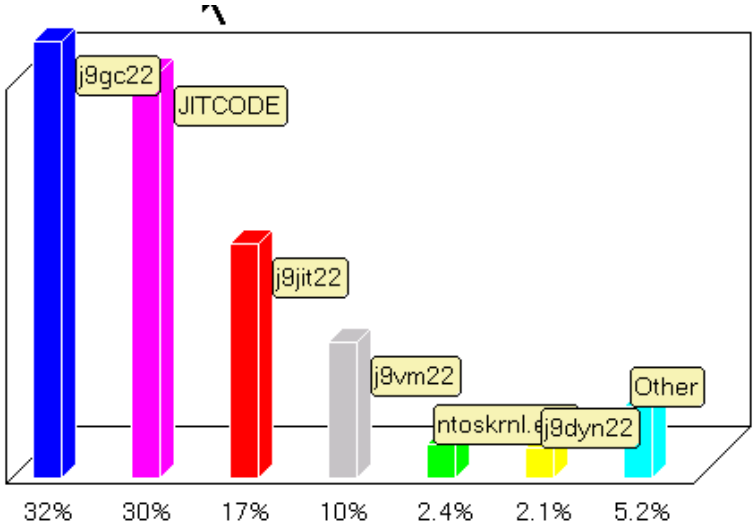
Note: If you cannot see this view from within the Profile Analyzer perspective, select **Windows -> Open view -> Other -> Profile Analyzer -> Samples Distribution Chart**.

This view provides a starting point for determining where you should focus your attention. For example, the following screenshot shows the graph for a java process using 23.4% of total profile time:



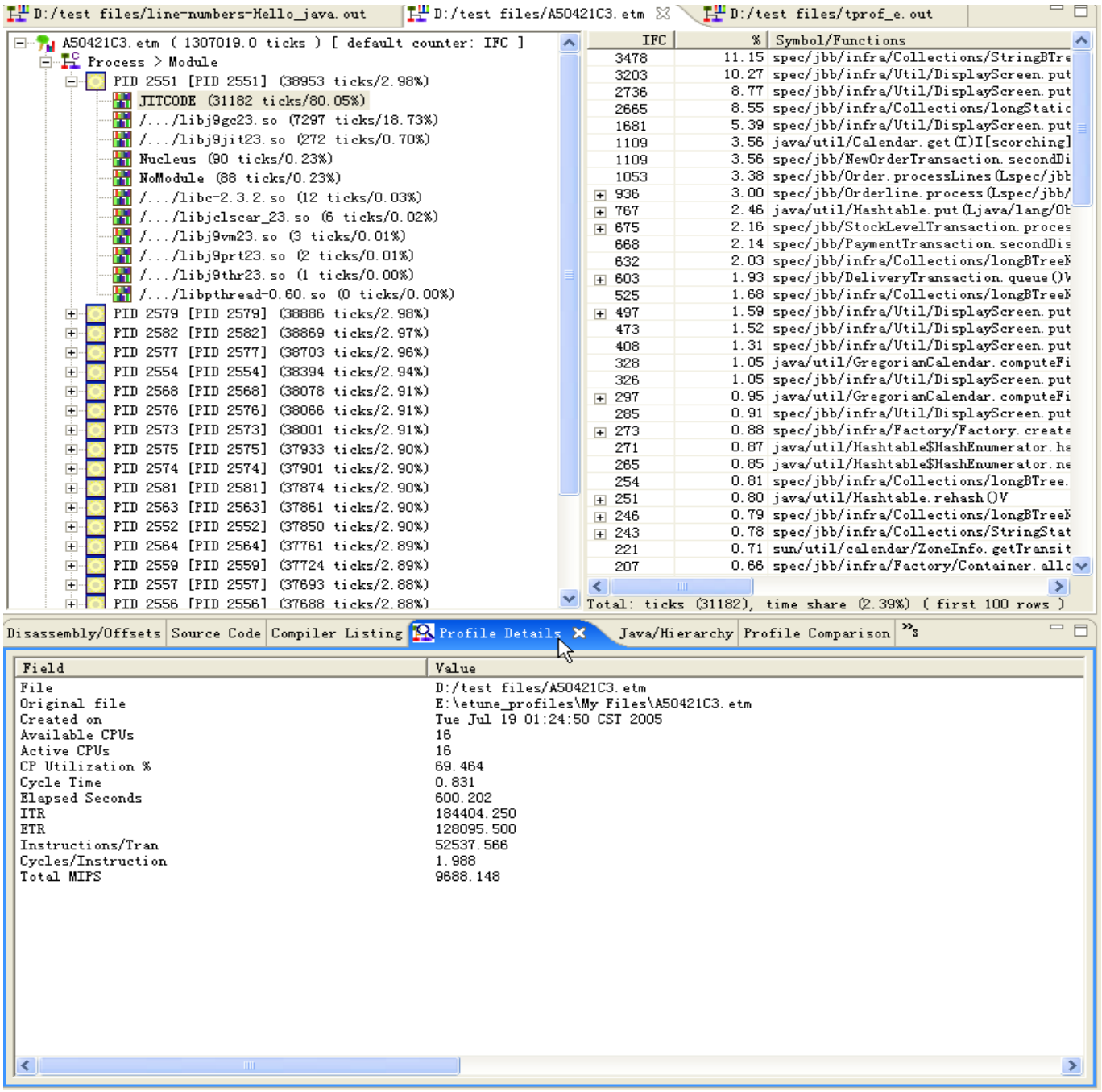
This graph shows that the JITCODE module (the module containing JIT-compiled Java methods) was the busiest module for this process, which suggests that some tuning of Java methods may be advisable.

The following graph (for a different Java program on a different system) shows heavy activity both in j9gc22 and in JITCODE. j9gc22 is the Garbage Collection library of the IBM Virtual Machine for Java (J9) indicating that the application may be memory constrained, or may be creating new objects too frequently. The third column (j9jit22) is the JIT compiler library, indicating that the profile may not have run for very long, since long-running applications typically have a small percentage of time used by the JIT. (A high percentage of time in the JIT may also indicate excessive use of invoke_interface calls, which require JIT library runtime support even when executing JITted methods).



5.1.4.7View profile details

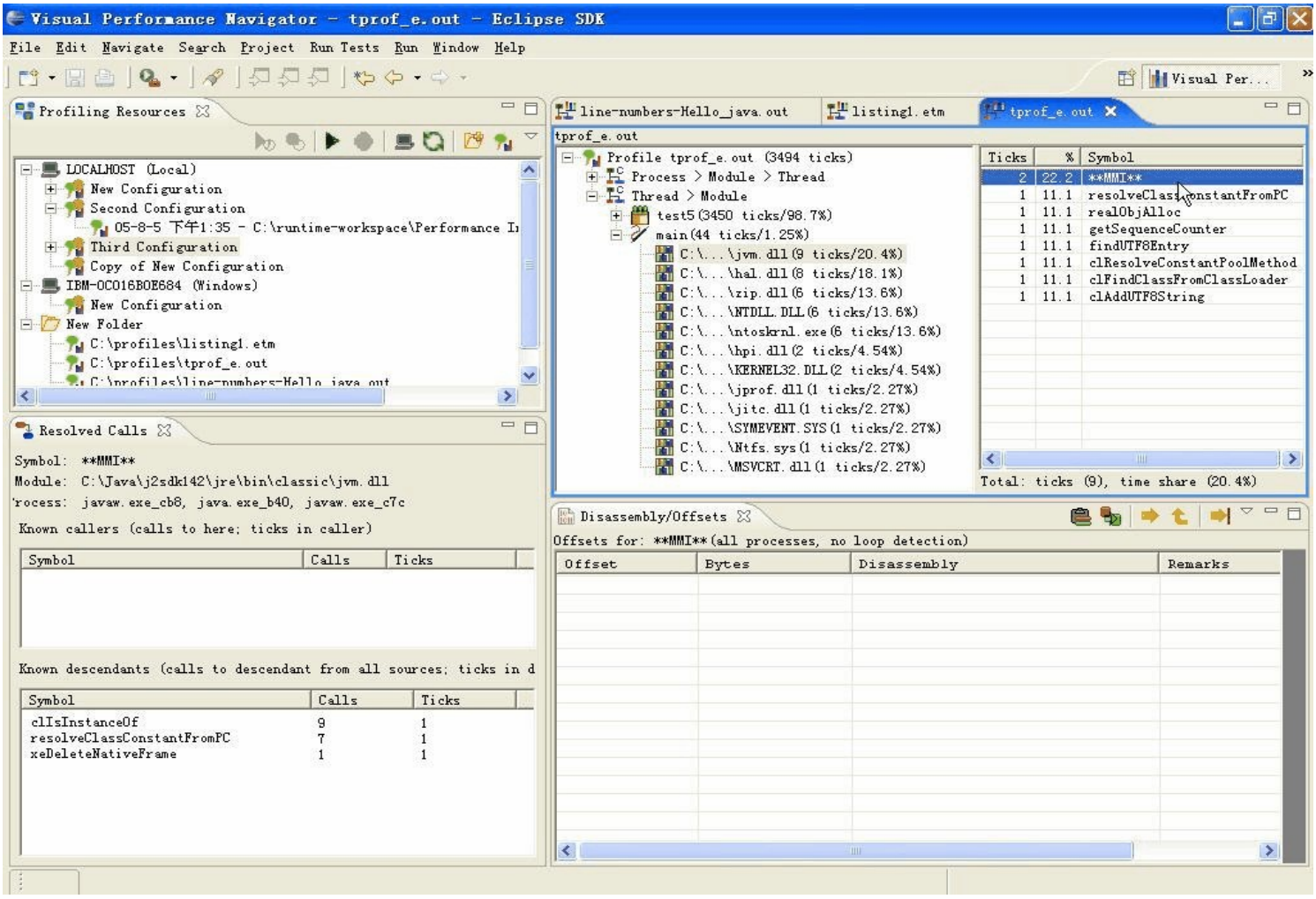
You can see the detail information of a profile file when you select it in the **Process Hierarchy View**. We can see it from the following example:



A file A50421C3.out is opened, and its corresponding information shows up in the **Profile Details** View.

5.1.4.8View resolved call information

When you open an IA32 profile, Profile Analyzer can analyze the disassembly in it to identify all call sites that have an immediate address as a target, and can attempt to connect those call sites to target symbols. This is done automatically for you if the **Resolved Call Information** view is visible. The following shows the **Resolved Call Information** view for a method selected from the jvm.dll module:

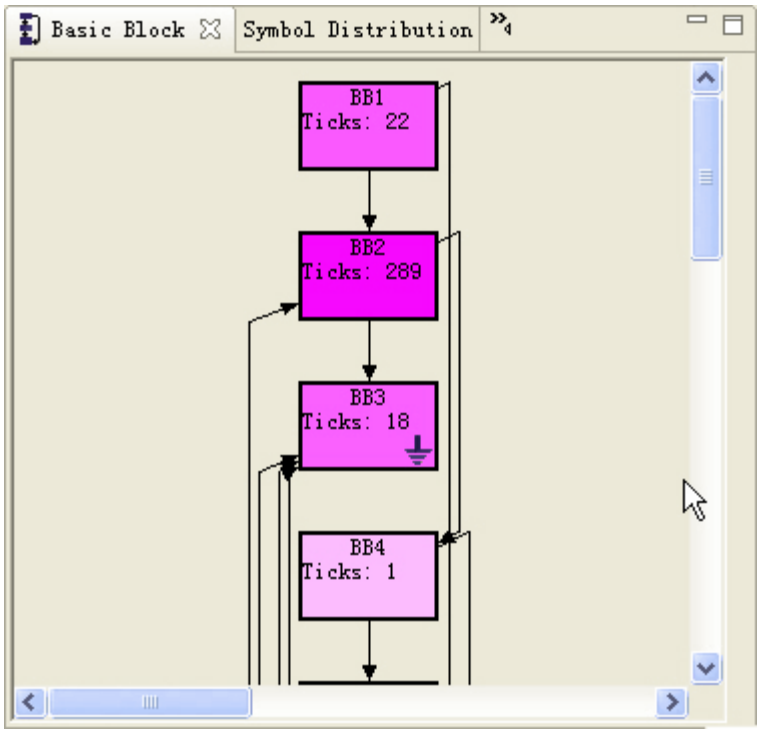


5.1.4.9View basic blocks

A basic block is a block of instructions that contain a single entry point and at most two exit points. Basic blocks are a concept used by compilers to perform dataflow analysis and to perform effective optimizations. Profile Analyzer attempts to detect basic blocks by analyzing the targets of all branch instructions within the disassembly for a symbol. Note that the basic blocks detected by Profile Analyzer may not match the basic blocks indicated in a compiler listing, as the compiler may use a higher-level basic block structure that includes internal branches. For example, a single source or intermediate-language instruction would likely not span multiple basic blocks from a compiler perspective. However, some source or intermediate-language instructions may result in multiple basic blocks at the disassembly level. An array assignment operation in Java is one such instance: the assignment is a single source statement, but may require both a null check and an array bounds check, each of which are intermediate-language instructions that may result in multiple conditional branches in the resulting disassembly.

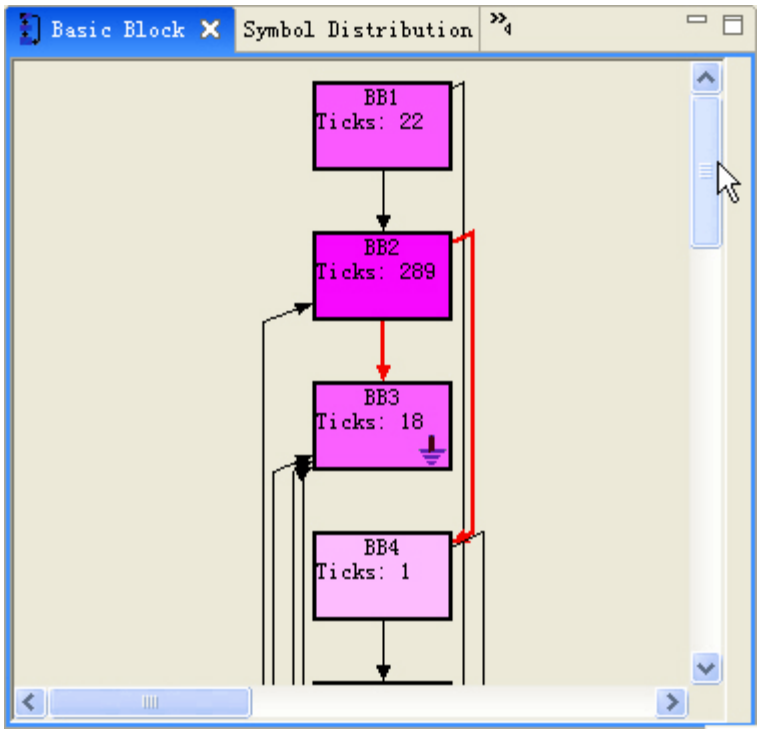
You can see basic block information by choosing **Windows-> Show View -> Other... -> Profile Analyzer -> Basic Block** or just find it in the left panes.

When you open a process, you can see its basic block information as follows:



Each basic block has a number (BB1, BB2 etc.), a tick count, zero or more incoming edges, and one or two outgoing edges (a terminating basic block does not have any outgoing edges). Each block with ticks is colored red, magenta or blue according to the same rules used to determine symbol tick color, and shaded according to the relative tick count of the basic block as compared to the symbol as a whole.

You can click on a basic block to highlight its incoming and outgoing edges in red:



In this view, BB2 was selected, and its outgoing edges to BB3 (the "fallthrough" basic block) and BB4 (the target basic block) are highlighted.

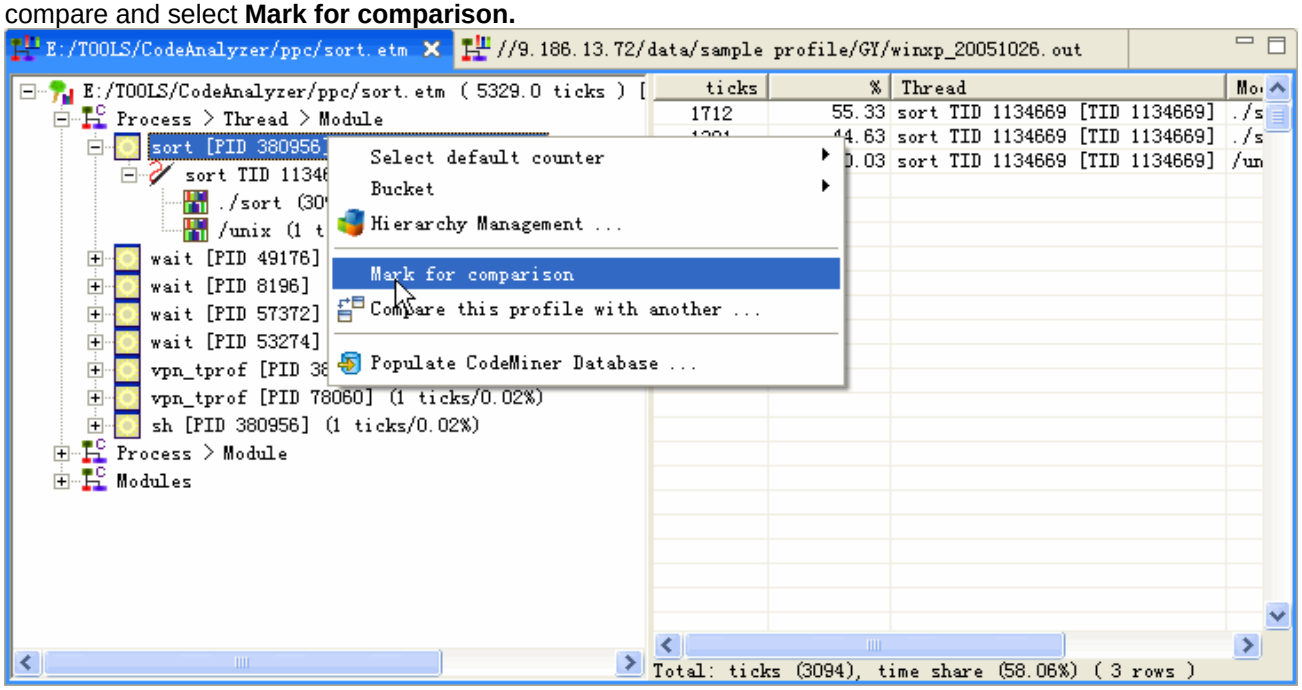
5.1.5Profile Comparison

The following are tasks you can do to compare profiles within Profile Analyzer:

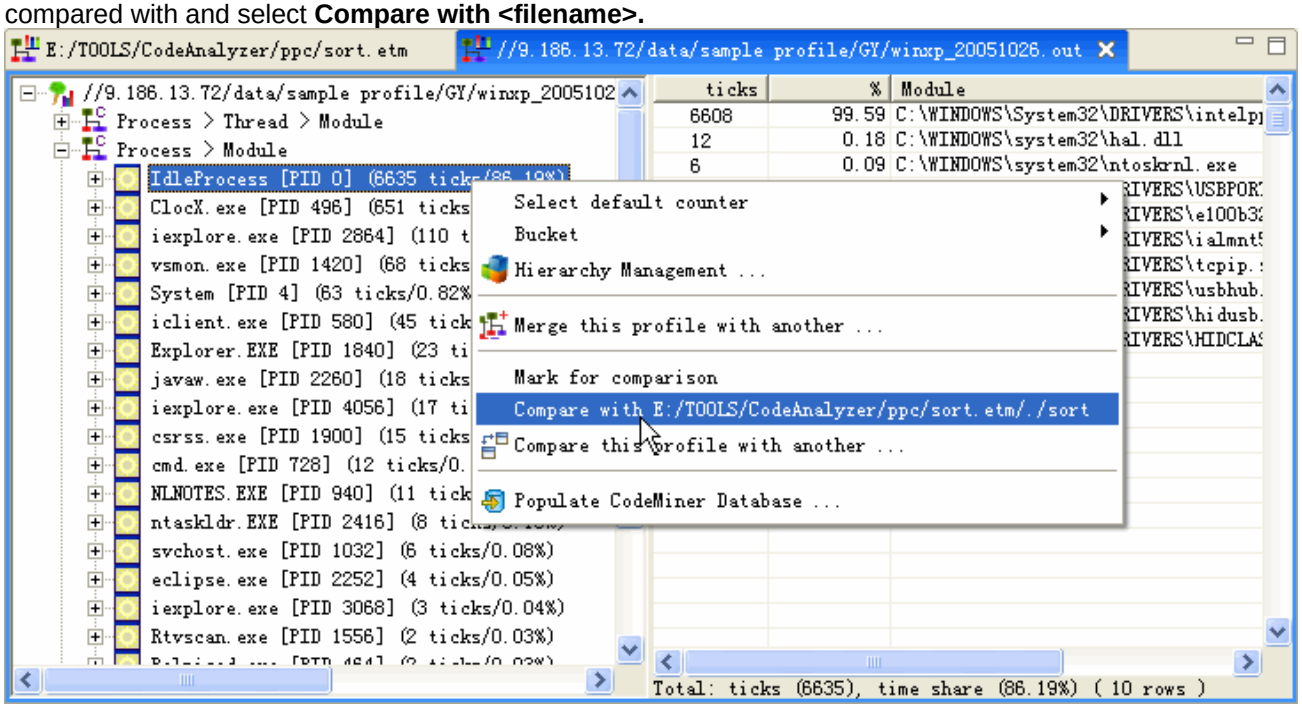
5.1.5.1Compare two profiles

1. Click the **Compare two profiles**  button in the **Profile Comparison** view toolbar. Alternatively, for any open profile, right-click in the process hierarchy view and select **Compare this profile with another**.
2. In the Profile Comparison Wizard, select two profiles you want to compare. The wizard supports compressed (.etz) profiles. Comparison of tprof profiles is still limited in that it does not expose compilation levels.
3. Click **Next**. Enter the following values for each profile:
    - **Transaction rate:** The transaction rate is the number of transactions completed per elapsed second. What we call a transaction varies from workload to workload, but it is a consistent indicator of the amount of work we are doing per second. If we are running background jobs, the number of transactions might be the number of jobs. If we are running an HTTP server, the number of transactions might be the number of HTTP requests we have served. For each type of workload, the transaction is clearly defined.
    - **CPU utilization:** CPU utilization is a percentage that describes how busy servers are. It is defined as the average utilization of all CPUs in a server.
    - **Number of CPUs:** This is the number of CPUs available to the system in each of the profile runs being compared. It is very common that we compare runs with different number of CPUs. For example, we may compare a 1-CPU run against a 4-CPU run to determine how well a workload scales up as we add CPUs to the configuration.
4. Click **Finish**. Two files are loaded and opened.

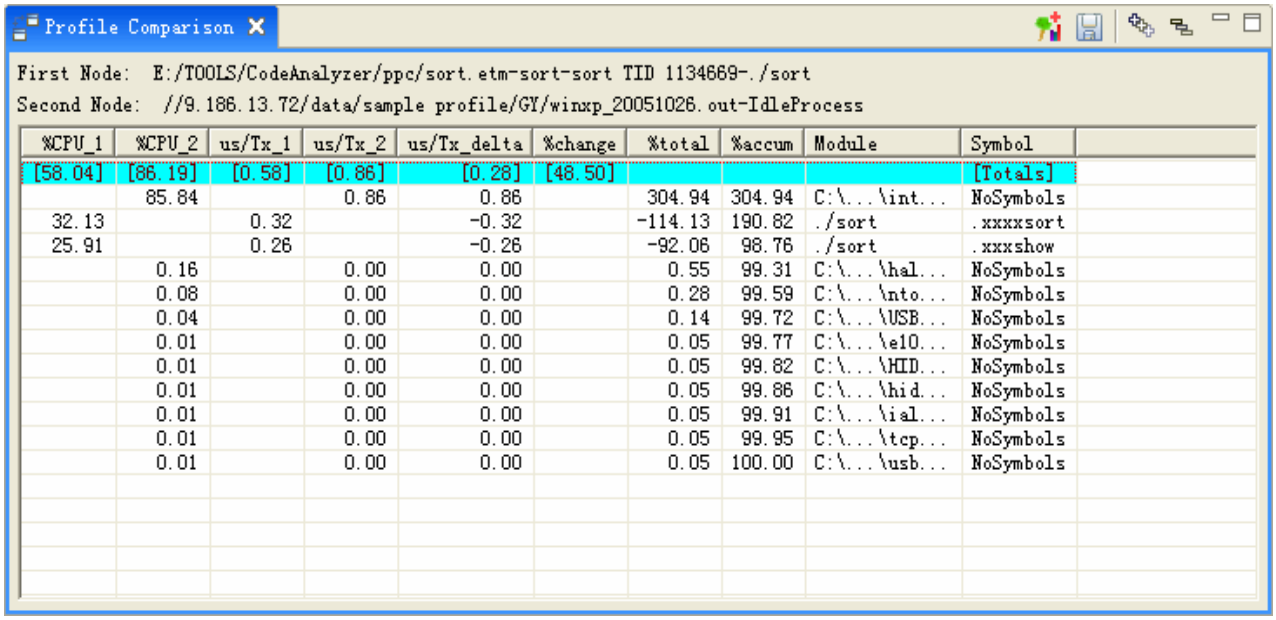
5. Right-click the process, thread or module of the first file in the process hierarchy view which you want to compare and select **Mark for comparison.**



6. Go to the other profile, right-click the process, thread of module of the second file which you want to be compared with and select **Compare with <filename>.**



7. In the **Profile Comparison** view, the detailed information of modules you have selected to compare is listed as follows:



%CPU_1	%CPU_2	us/Tx_1	us/Tx_2	us/Tx_delta	%change	%total	%accum	Module	Symbol
[58.04]	[86.19]	[0.58]	[0.86]	[0.28]	[48.50]				[Totals]
	85.84		0.86	0.86		304.94	304.94	C:\...\int...	NoSymbols
32.13		0.32		-0.32		-114.13	190.82	./sort	.xxxxsort
25.91		0.26		-0.26		-92.06	98.76	./sort	.xxxshow
	0.16		0.00	0.00		0.55	99.31	C:\...\hal...	NoSymbols
	0.08		0.00	0.00		0.28	99.59	C:\...\nto...	NoSymbols
	0.04		0.00	0.00		0.14	99.72	C:\...\USB...	NoSymbols
	0.01		0.00	0.00		0.05	99.77	C:\...\e10...	NoSymbols
	0.01		0.00	0.00		0.05	99.82	C:\...\HID...	NoSymbols
	0.01		0.00	0.00		0.05	99.86	C:\...\hid...	NoSymbols
	0.01		0.00	0.00		0.05	99.91	C:\...\ial...	NoSymbols
	0.01		0.00	0.00		0.05	99.95	C:\...\tcp...	NoSymbols
	0.01		0.00	0.00		0.05	100.00	C:\...\usb...	NoSymbols

5.1.5.2Understand the calculations

The comparison tool uses the normalization factors entered in the Profile Comparison Wizard to calculate the microseconds of CPU consumed per transaction (us/Tx). Since the us/Tx values are computed on a per transaction basis, they can be compared directly from profile to profile.

The us/Tx values are calculated as follows:

1. *Calculate percentage of total ticks in the specific symbol:*
   CPU% = Ticks in the specific symbol / Total ticks in the profile run

2. *Calculate transactions per busy second:*
   ITR = Transaction rate / CPU utilization

3. *Calculate total CPU microseconds per transaction:*
   Total microseconds per transaction = 1,000,000 / ITR * Number of CPUs

4. *Calculate average CPU microseconds per transaction in the specific symbol:*
   us/Tx = Total microseconds per transaction * CPU%

5.1.5.3Save a profile comparison

1. Click the **Save Comparison** button in the Comparison view toolbar. Alternatively, right-click anywhere in the view and select **Save Comparison** from the pop-up menu.

2. The **Save As** dialog opens. Browse to the desired directory and enter a file name.

3. Click **Save**. The comparison will be saved as a Profile Analyzer comparison (.etc) file. This file contains both compared profiles (zipped) and the normalization factors used in the comparison.

5.1.5.4Open a profile comparison

1. In the Navigator view, double-click the Profile Analyzer comparison (.etc) file.
2. Both compared profiles will open automatically in the Profile Analyzer editor as temporary files, and the Comparison view populated.

5.1.6Profile Merge

If user has profiled a benchmark multiple times using TPROF, he can merge the .out files for these runs using the Merge Wizard in Profile Analyzer. This can be useful for several types of situations:

- If user is going to measure a short-run application (or a short-run phase such as startup of a JVM during a benchmark), each individual profile may have too few ticks to draw meaningful results, but a with a merged profile patterns may begin to emerge
- If user is going to measure different CPU events (ticks, data cache misses, branch mispredictions) on different runs of a benchmark, he can merge these runs and see the data for all counters in a single profile
- If user wants to compare two runs, he can use profile merging to see which processes, modules, symbols, and symbol offsets were active in both runs.

To merge several profiles, at least one of them must be opened. Then follow these steps:

1. Right-click in the process hierarchy view and choose **Merge this profile with another**.

2. In the **Merge Profiles Wizard**, select one or more profiles from the current project and click **Add >**, or use the **Browse** button to open a file dialog to select profiles from other locations. Note that Profile Analyzer will only let user to add profiles whose platform matches the profile already added to the list. Click **Next**.

3. On the **Select processor type from platform family** page, select a CPU type (this option is only available on platforms where different CPU types support different counters). Click **Next**.

4. On the **Counter options** page, user can select each profile in turn and chooses what counter to attribute its events to. One of the profiles must be the "primary" profile; this is the profile used to merge other profiles into. Click **Next**.

5. Select a file name and click **Finish**.

At any time the **Finish** button is not grayed out user can click it to merge immediately.

When user merges profiles, Profile Analyzer creates a new file with a .etm extension (ETM=e-Tune Merged). This file is in a Profile Analyzer-supported XML format. Profile Analyzer also opens the file immediately after the merge. This profile looks much like ordinary TPROF profiles when viewed inside Profile Analyzer, with three differences:

- Processes, modules, symbols, and offsets that had data from more than one source profile are colored in green
- Multiple counter columns may appear in the Offsets view as well as ticks, if user chooses different counters for each source profile.
- No threads data is available, as it does not make sense to merge thread data from separate runs.

5.1.7Symbol Analysis

The following are tasks that you can perform to analyze symbols within Profile Analyzer:

5.1.7.1Code Miner support

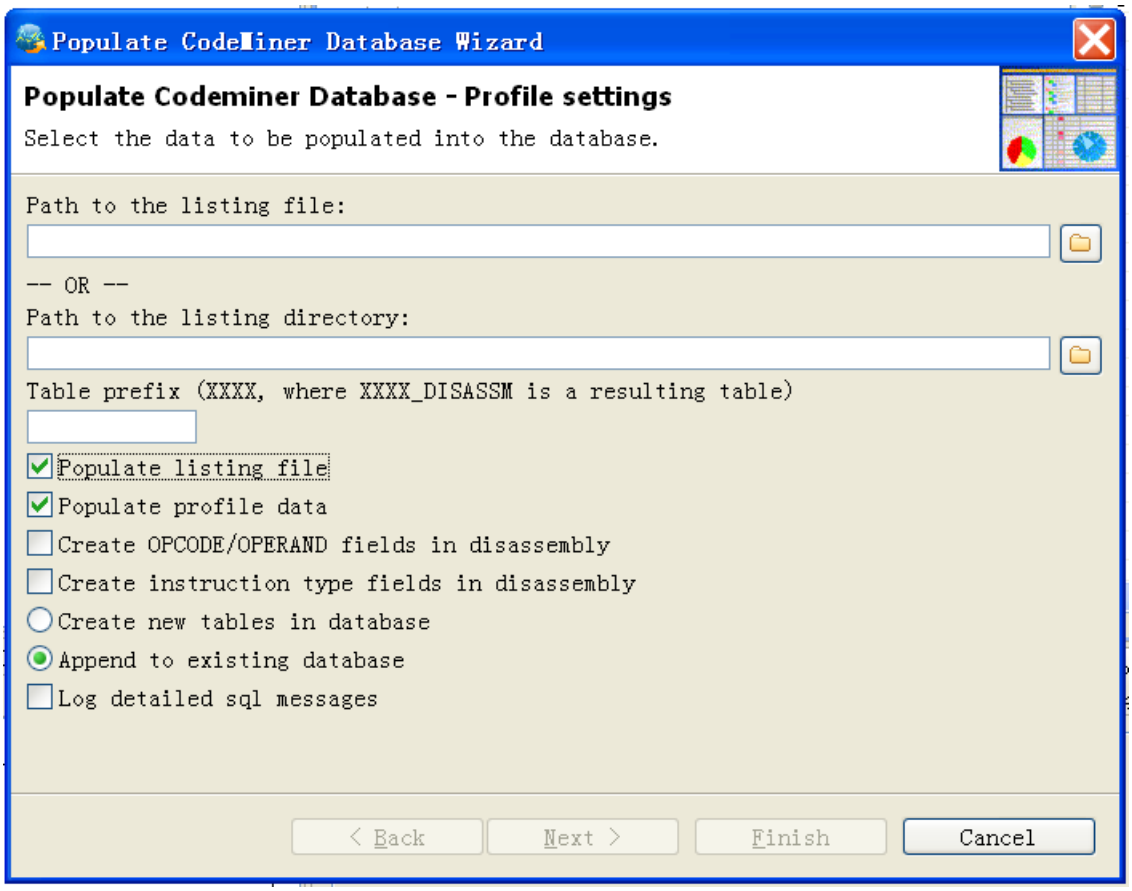
The Code Miner support in Visual Performance Analyzer enables you to populate an SQL database with information from Profile Analyzer profiles, and then perform SQL queries on the database to detect performance patterns that are not easily detected by traditional profilers. The database tables allow you to associate profile counter information, symbols, and disassembly instructions so that you can find inefficient or highly active patterns of instructions, instruction sequences, symbols, register usage, and so on. For example, a Code Miner query can be used to find the hottest pairs of sequential instructions, or all symbols that contain a particular instruction sequence, or all symbols that are hotter than a certain threshold that have a certain pattern in their name. Code Miner is ideal for analyzing flat profiles, namely profiles where no single symbol uses more than a fractional percentage of total ticks. In flat profiles, the objective is to find patterns of disassembly code, or usage patterns of certain types of symbols, that are inefficient. Without Code Miner it is extremely difficult to determine which patterns are worth investigating. Because Code Miner lets you determine the overall cost of a particular pattern within an entire module or an entire profile, you can use it to detect the patterns that will yield the maximum benefit when optimized, replaced, or eliminated.

Code Miner user interface support within Profile Analyzer includes a **Code Miner wizard** for populating data from a profile, and two views: a **Code Miner Query** view, that lets you query the Code Miner tables for a particular profile to find patterns of interest, and that displays the results in a sortable column-based table; and a **Query Tree** view that saves queries and database configurations so that you can easily locate, edit, re-run past queries or queries imported from another user, such as Compiler listings.

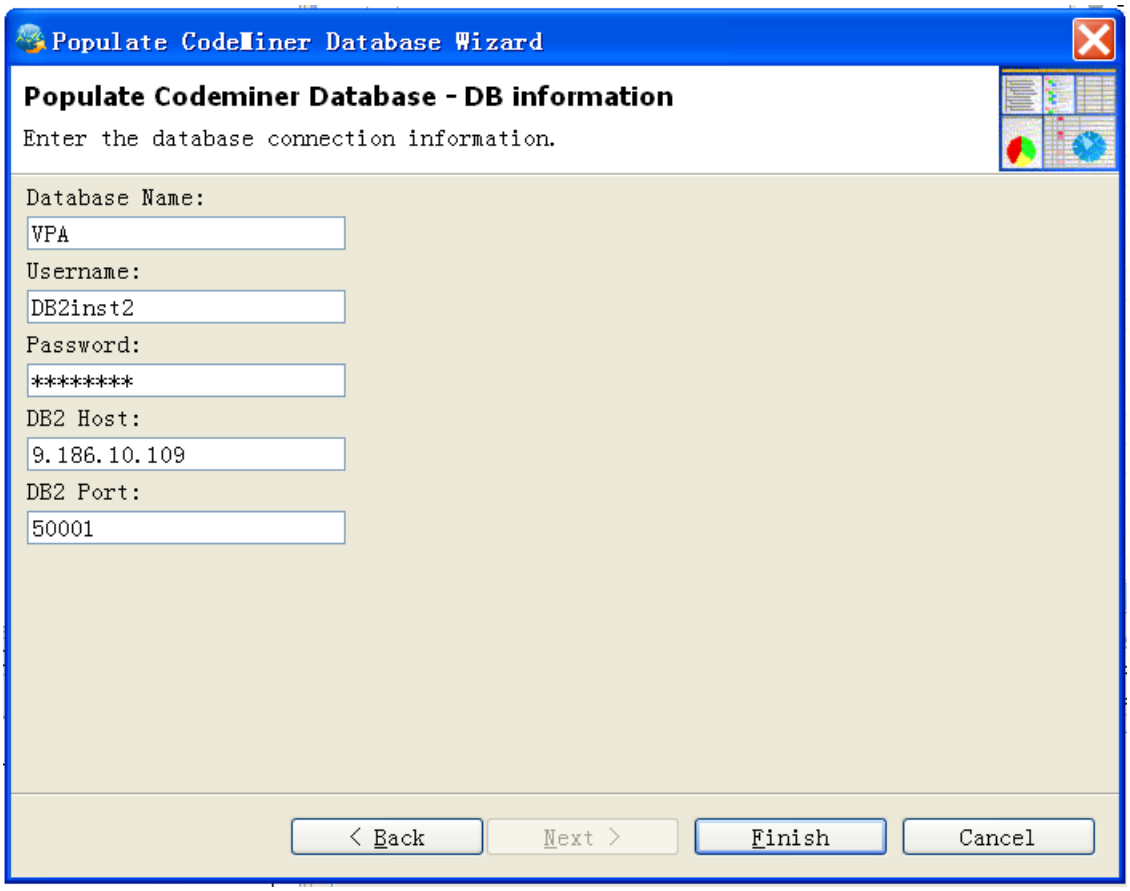
5.1.7.1.1Populate Code Miner Database

In Profile Analyzer, you can choose to keep profile or trace file data into DB2 database. This can be realized via **Populate Code Miner Database Wizard**. Every time when you open **Populate Code Miner Database Wizard**, you can choose to create new table, append to existing table or clean tables. If you decide to keep data into database, the prefix of table which is designed to store data should be defined at first. In the next page of wizard, given name, host, port and admin password, Profile Analyzer can get access to the database and populate data automatically. Please follow the steps below to populate profile file into new table in existing database:

1. Open a profile file and right click.
2. Choose **Populate CodeMiner Database.**
3. Defines table prefix, create new tables on database and include proper fields as you need.



4. Input db2 connection information into next wizard page. Be sure to pass firewall so as to connect database.



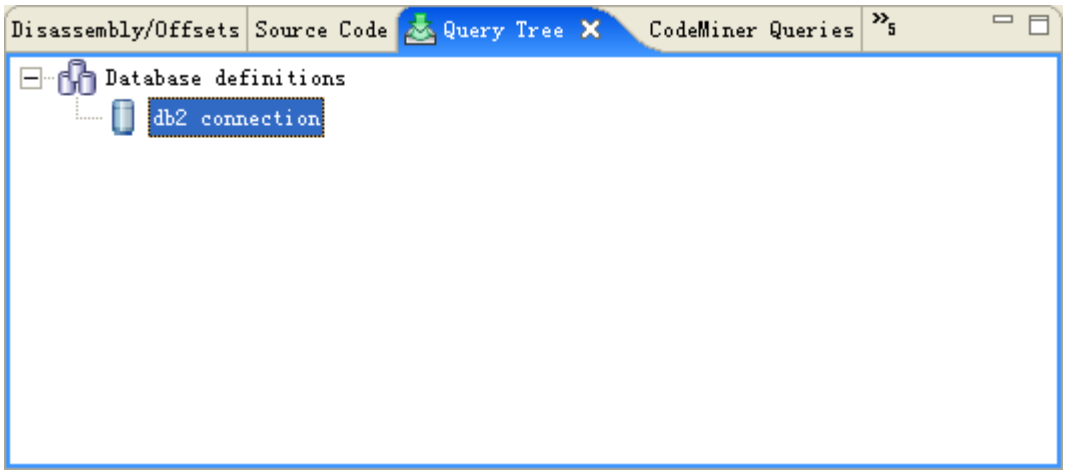
5. Click **Finish**

5.1.7.1.2Code Miner Database Queries

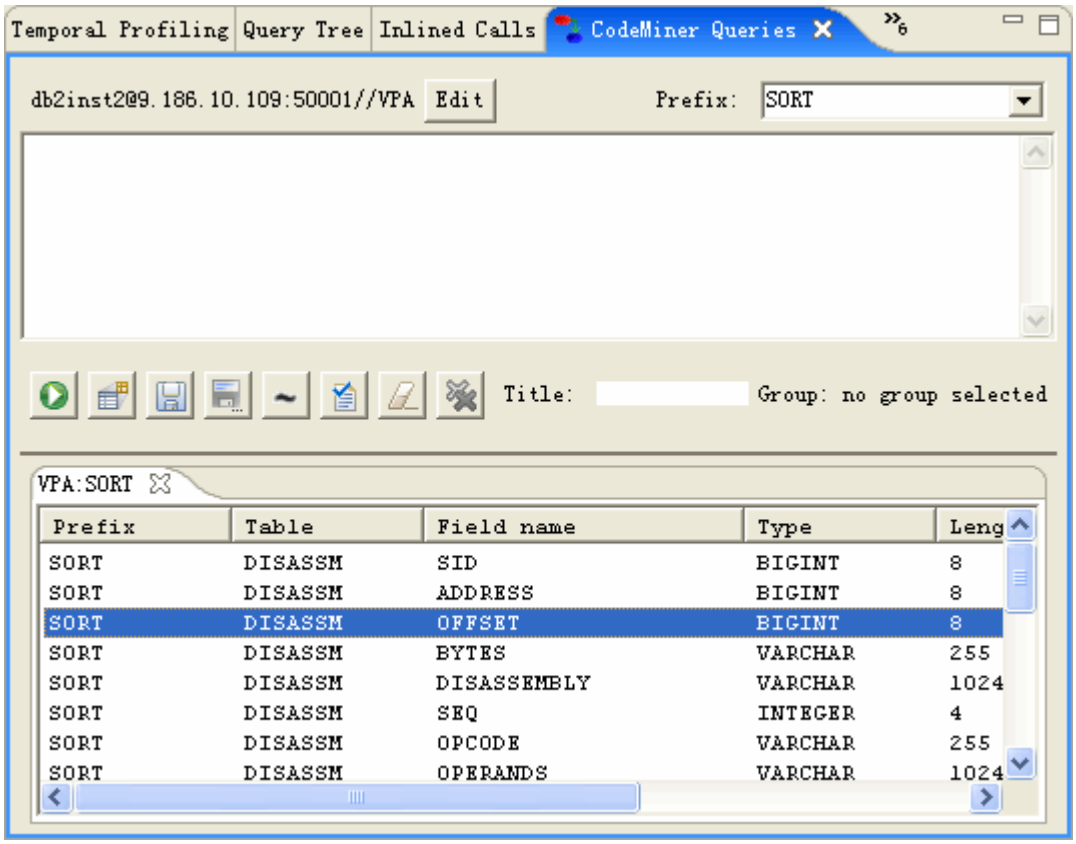
To check data in database, run query in **Code Miner Queries**. Please follow the steps below:

1. Create a new connection to the db2 database above through **Code Miner Queries** or **Query Tree** view

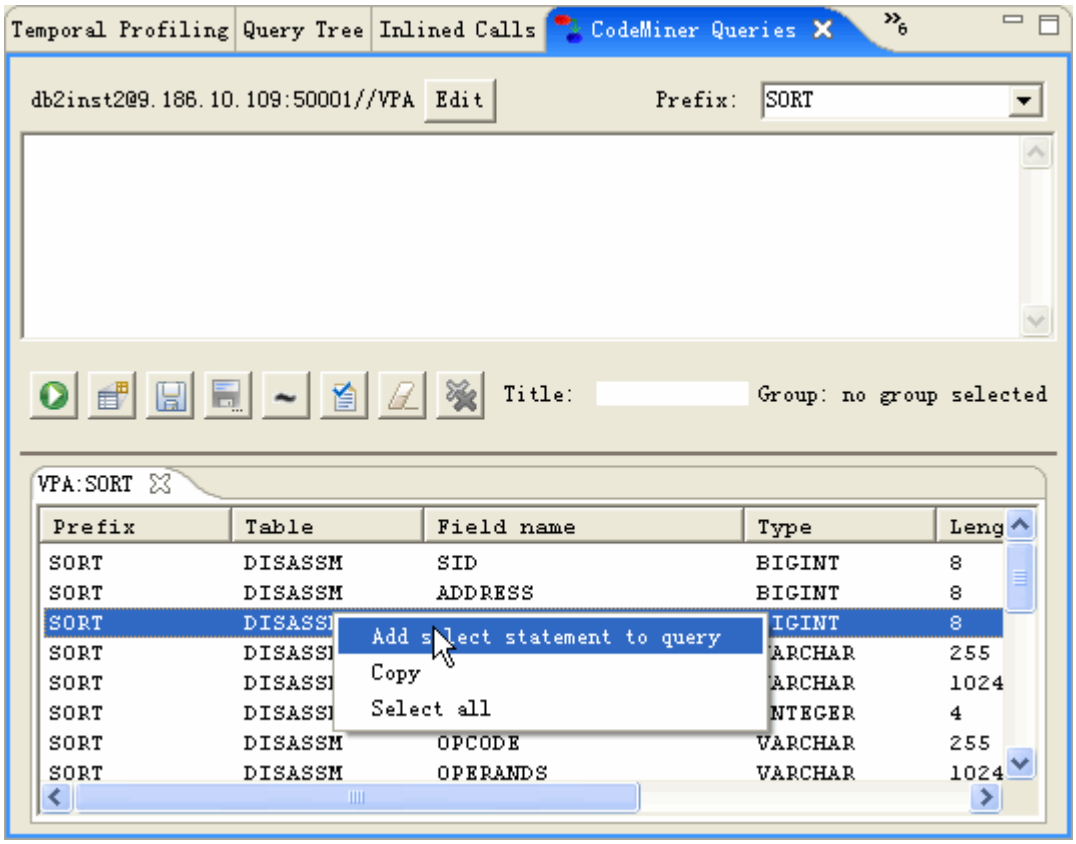
- Press **Edit** button in **Code Miner Queries** view
- Right-click and choose **Define Database Connection**



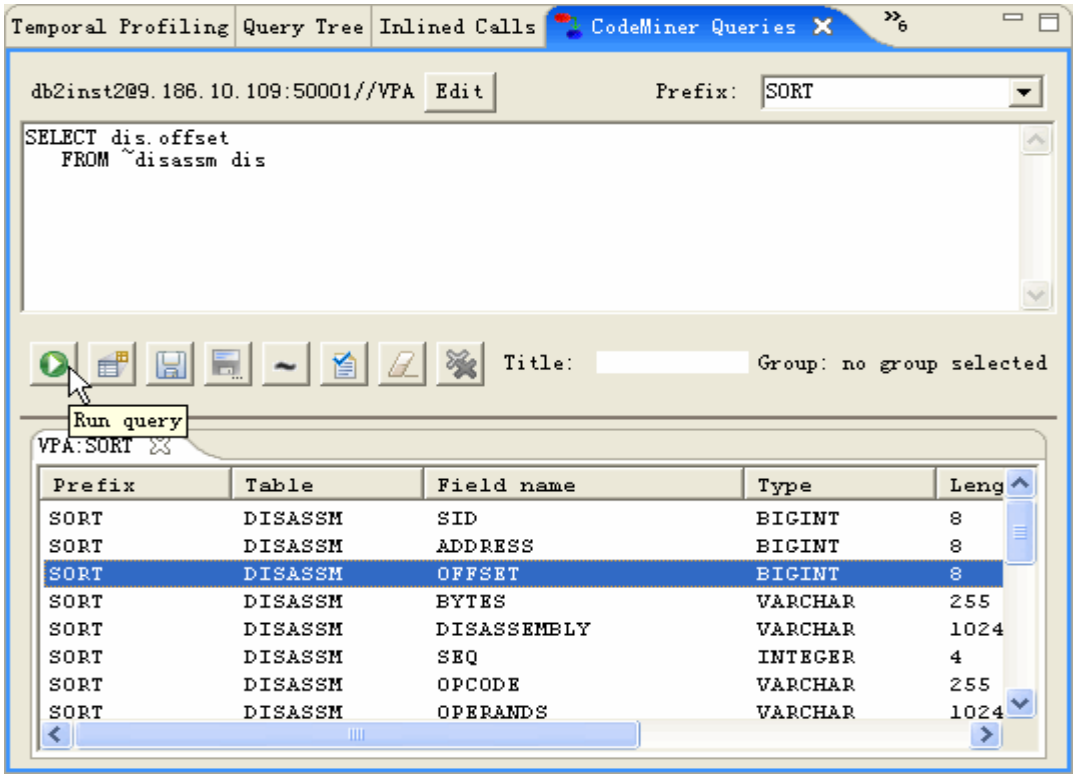
2. Input sort in Prefix box and press to list fields and tables with input prefix



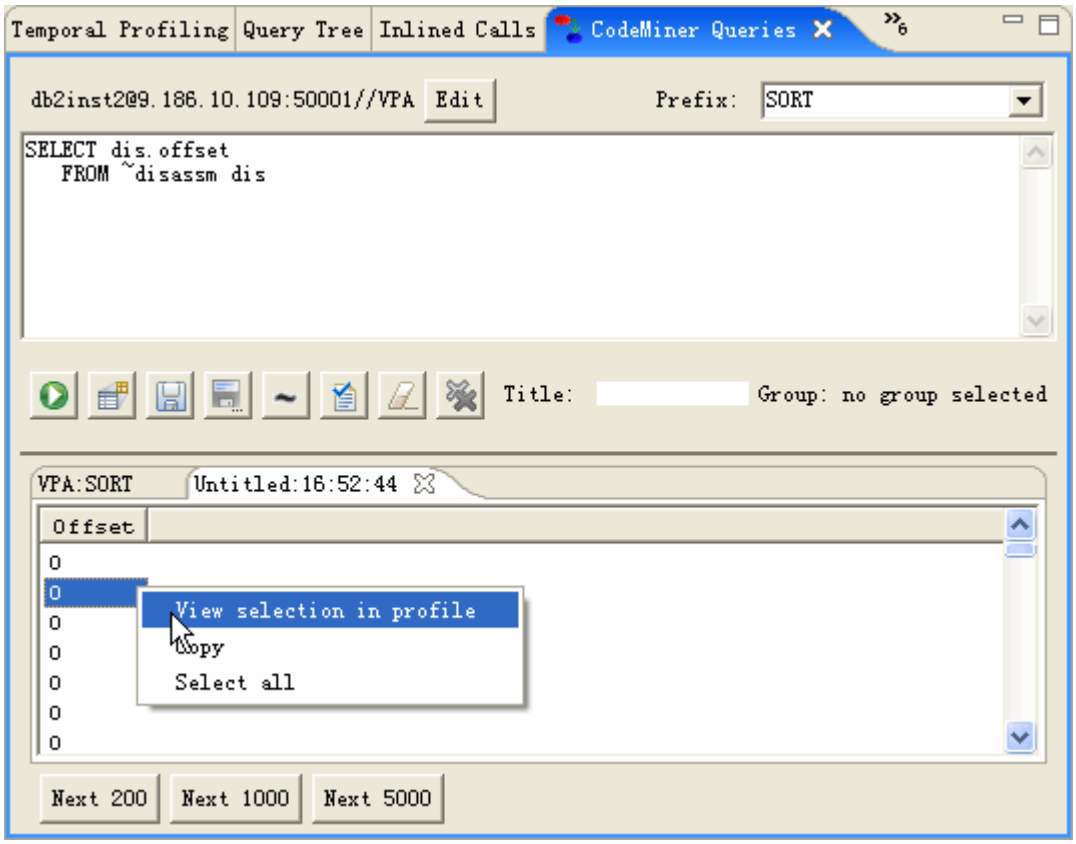
3. Choose lines and right-click to add select statement to query



4. Press  to run this statement in database



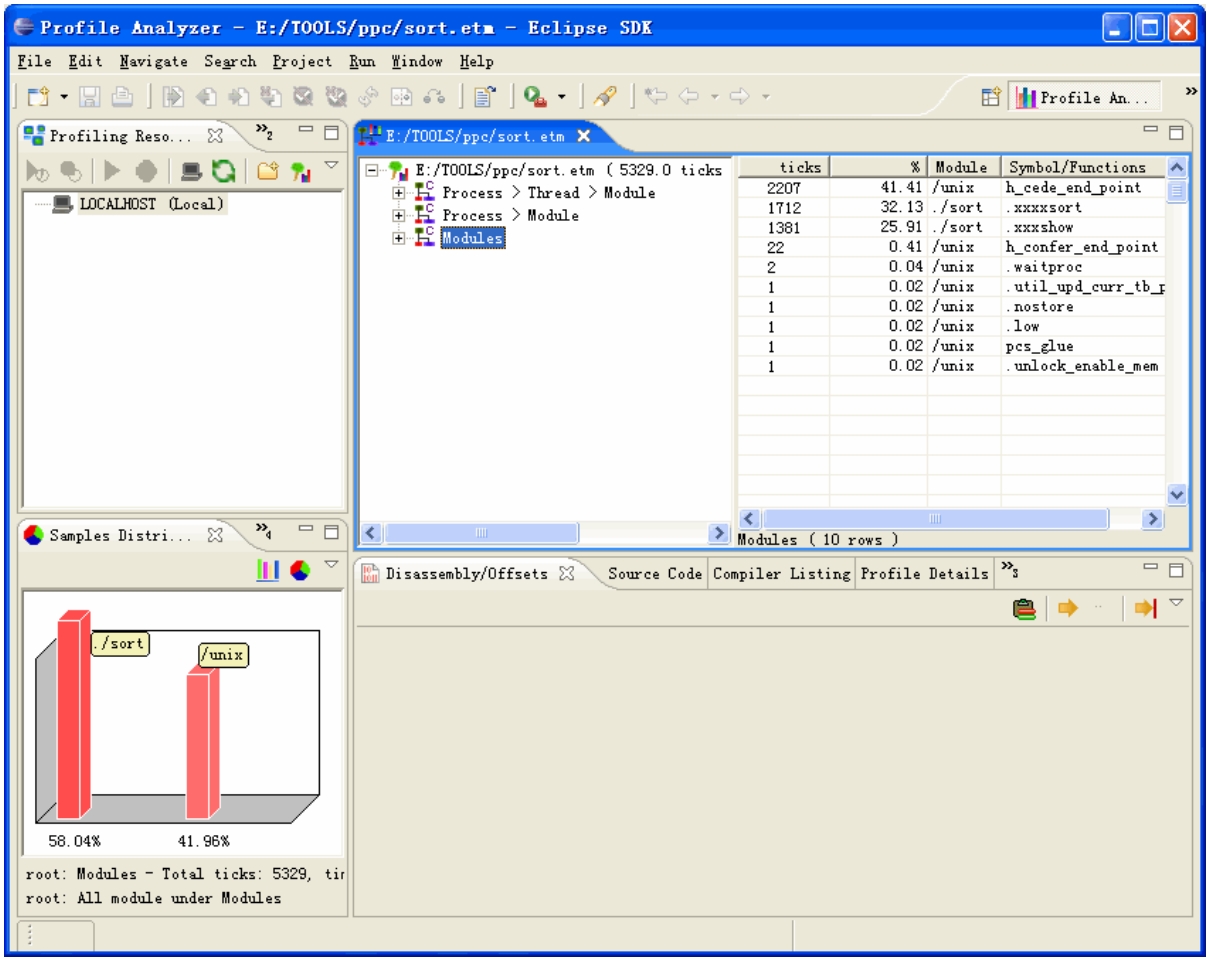
5. Double-click instruction item in new pop-up tab to view the profile this instruction belongs to



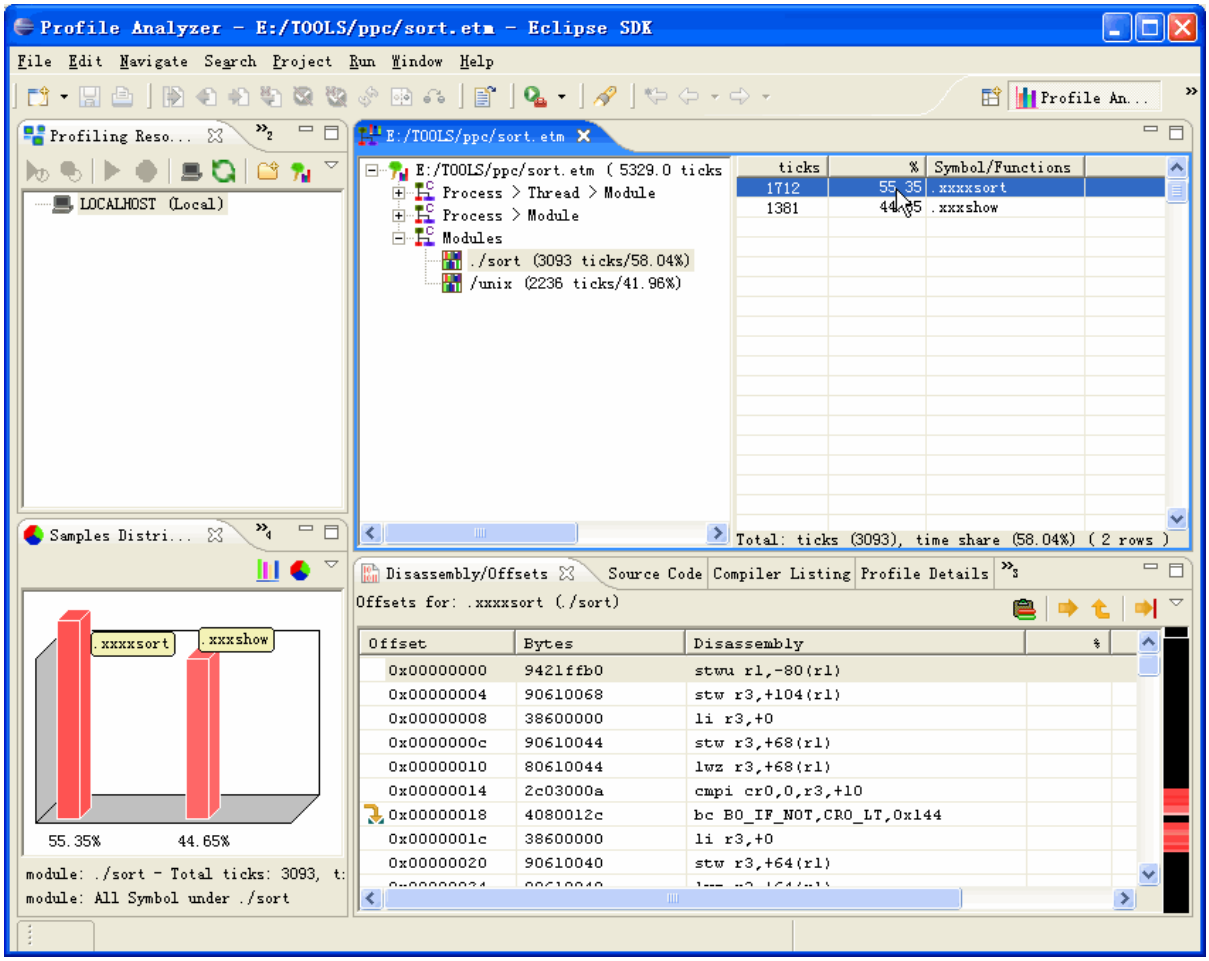
5.1.7.2Couple with Code Analyzer

Profile Analyzer can integrate with Code Analyzer for better navigation and comparison of module information between profiling file and executable file. This function can be initiated in generic hierarchy view. Then you can scroll both kind of information in Disassembly/Offsets view of Profile Analyzer and Instruction Table view of Code Analyzer at the same time. To couple with Code Analyzer, be sure to have both profile and binary file containing at least one same module. Please follow the steps below:

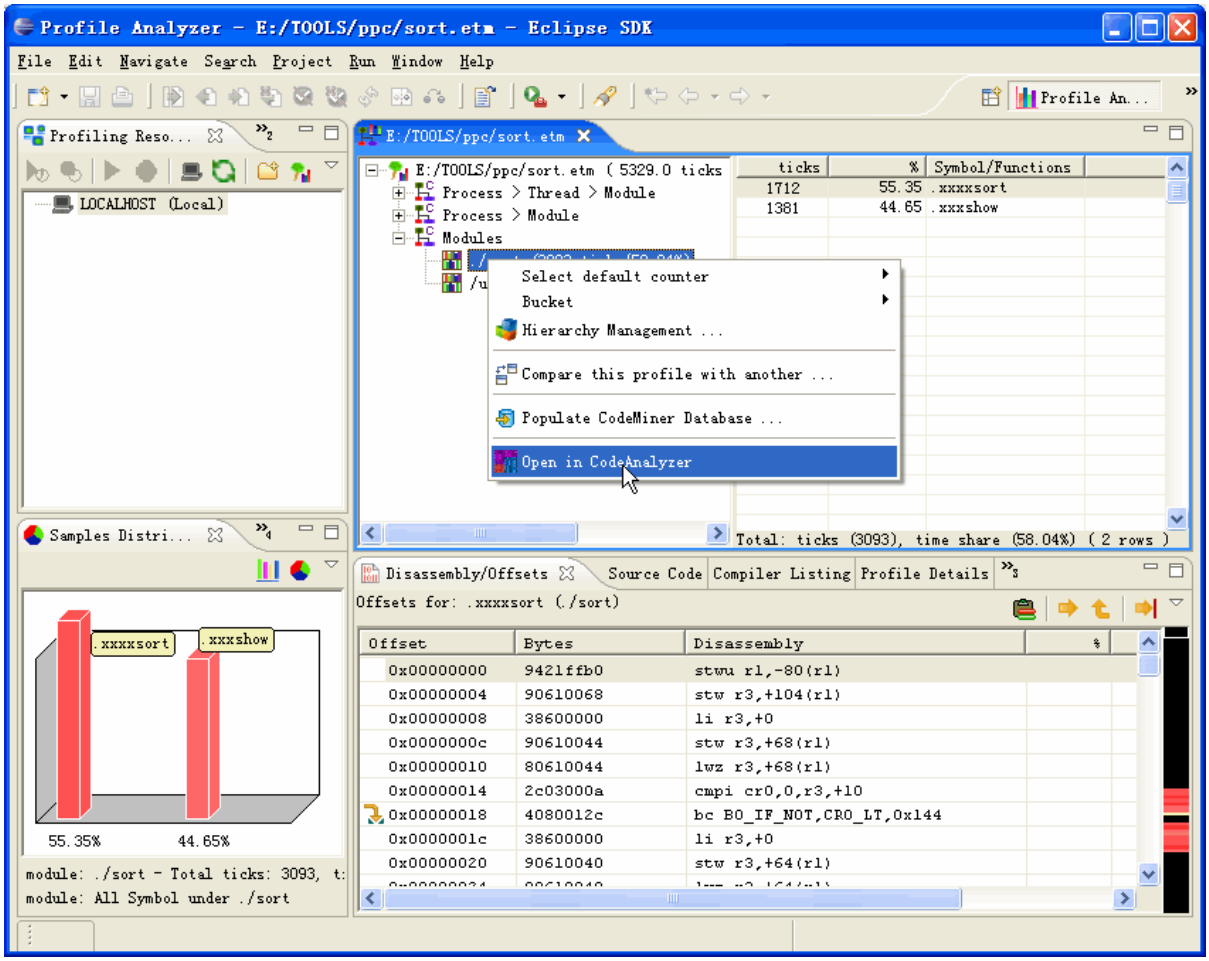
- Open a profile file.



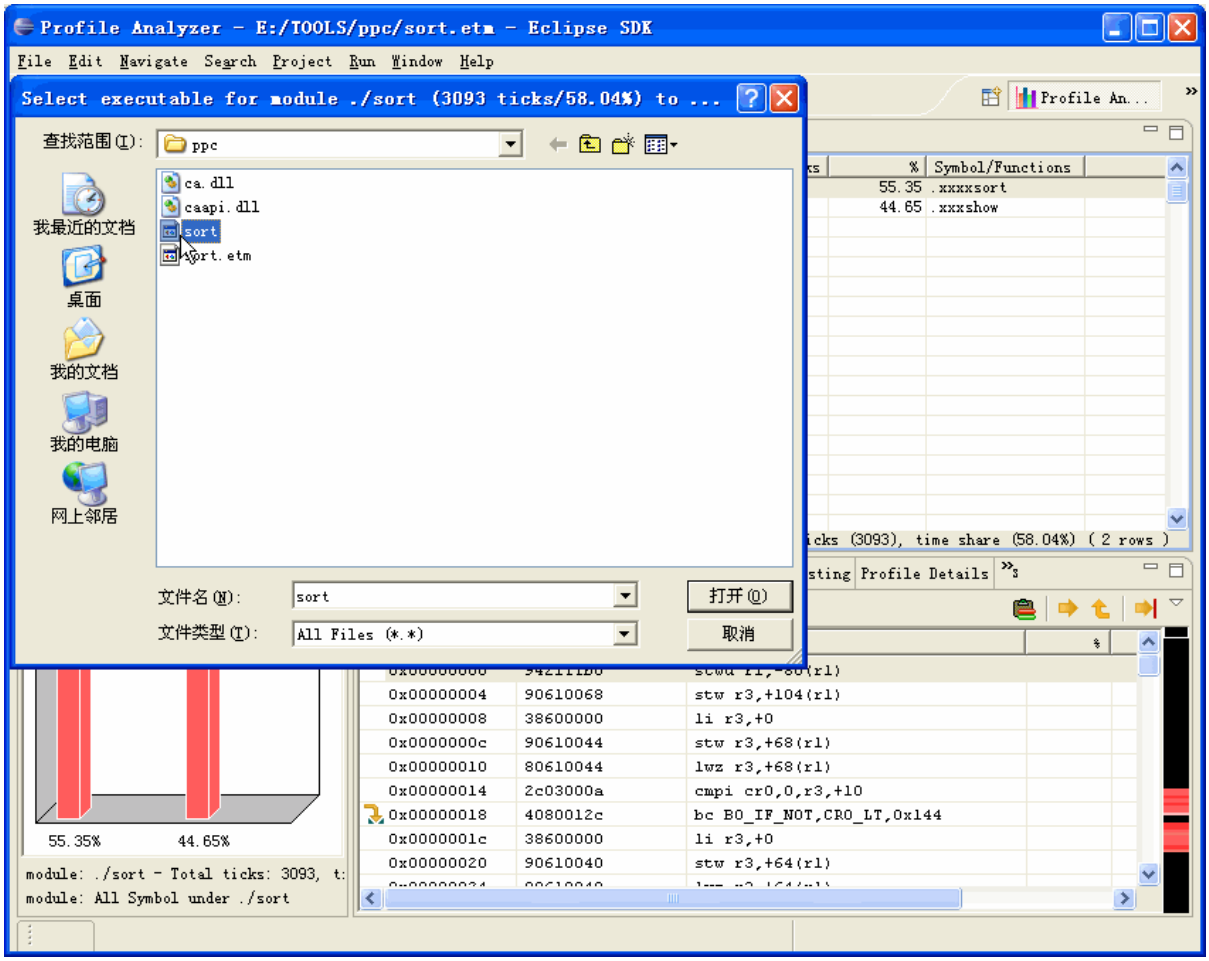
- Navigate generic hierarchy view, click a module and view its disassembly and offset information by pressing ENTER button.



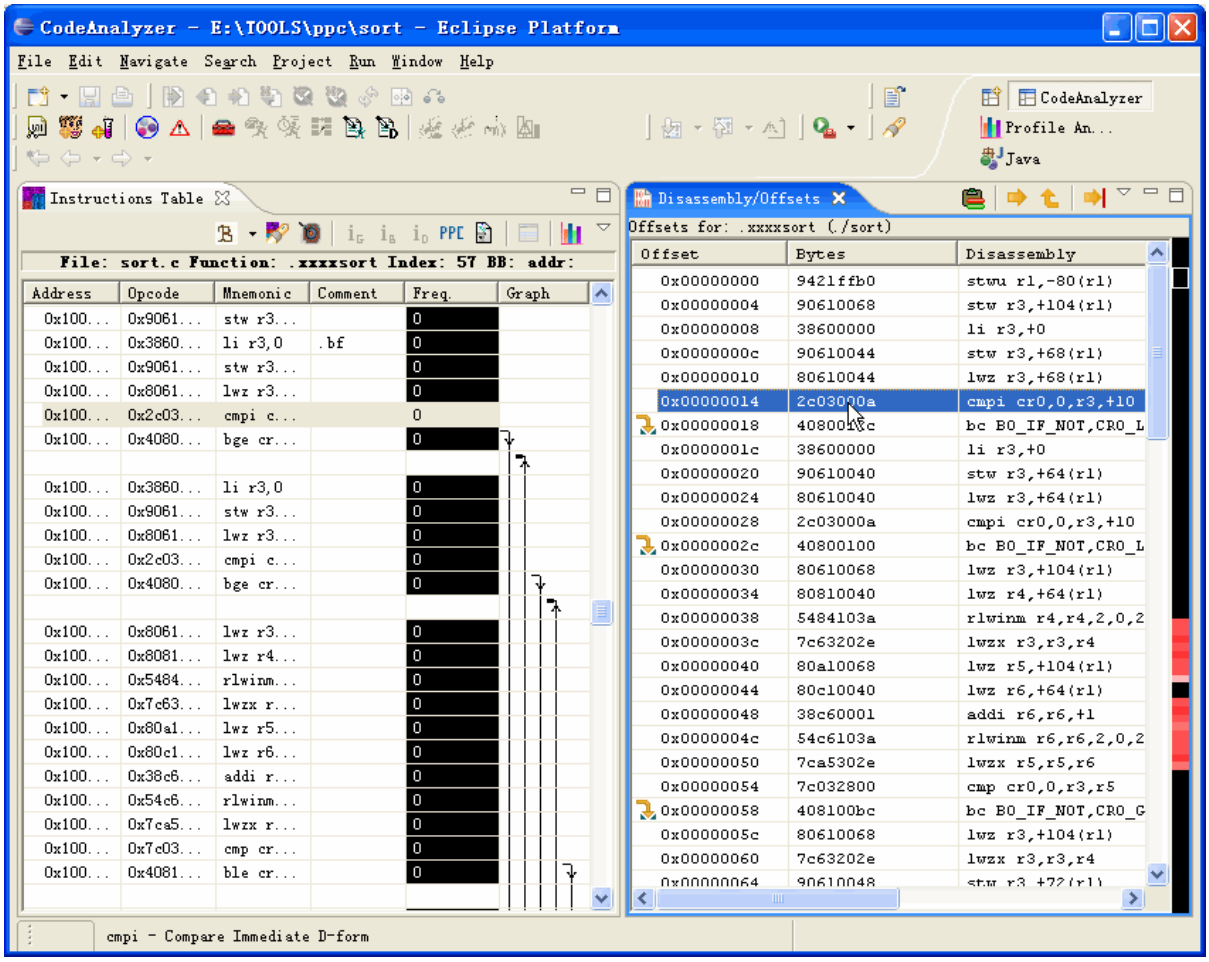
- Right-click this module symbol in generic hierarchy view. In popup menu, choose " Open in CodeAnalyzer".



- Choose the corresponding binary file of this module.



- Later, Code Analyzer Perspective opens automatically with this binary file. To scroll Profile Analyzer view with Code Analyzer view at the same time, be sure to open Disassembly/Offsets view. Now, when you select an table row in Disassembly/Offsets view, the instructions of this address will be highlighted in Instructions Table accordingly.



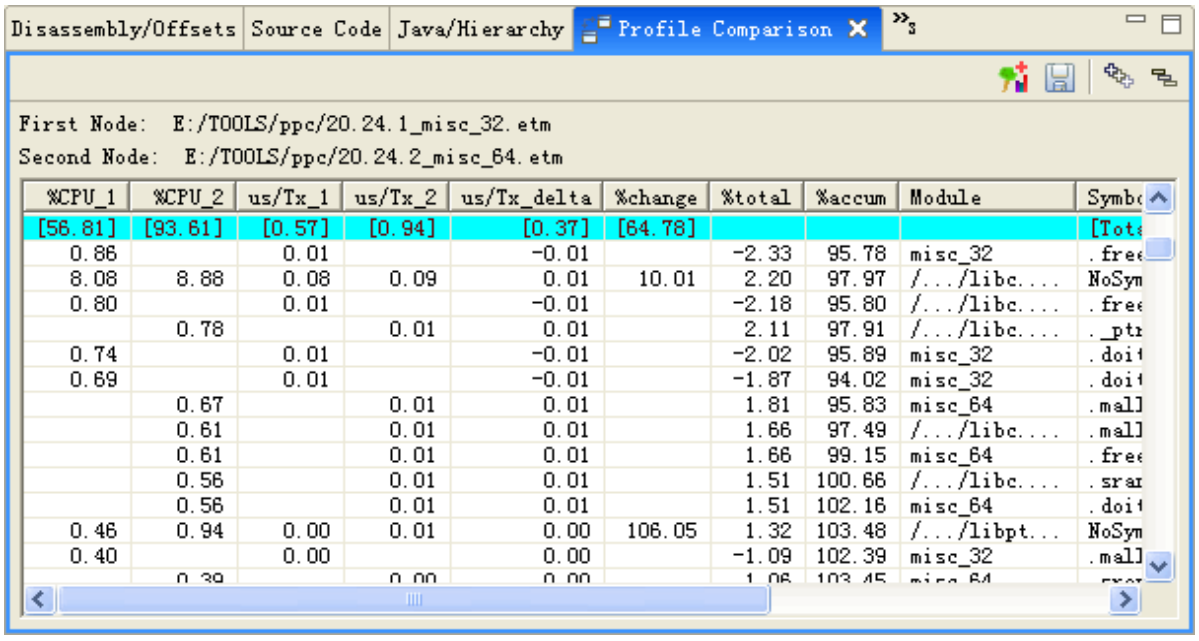
You may also trigger highlighting from Instructions Table of Code Analyzer. Be sure to press button in tool bar.

5.1.7.3View disassembly comparison

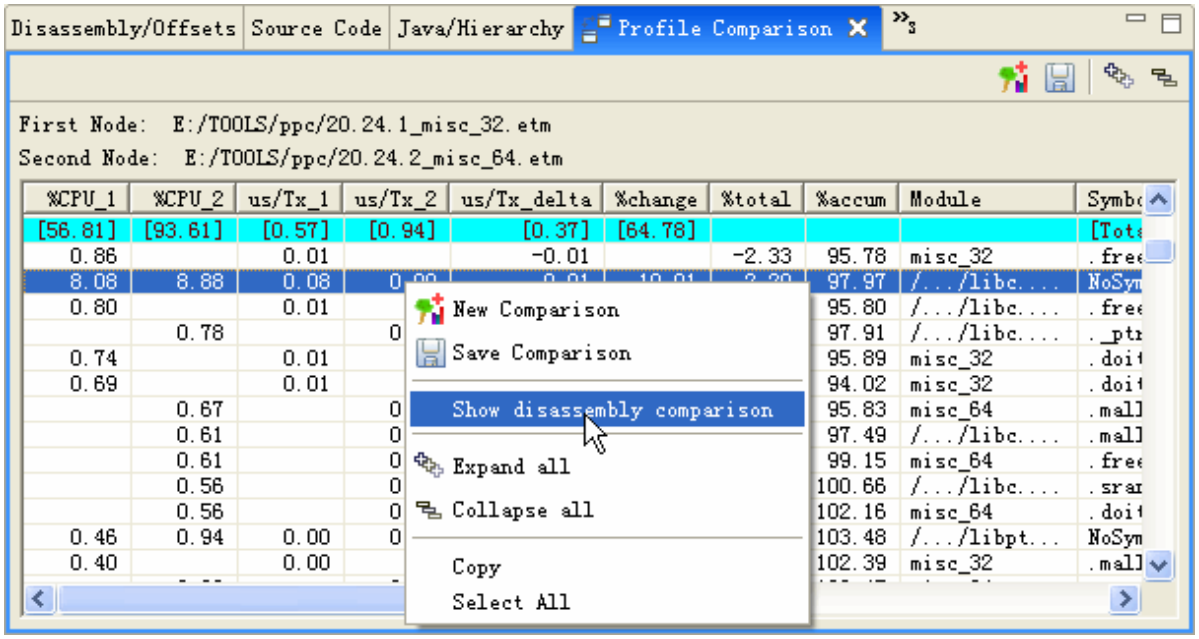
In this release, you can compare the offset, ticks and disassembly of two profiles in disassembly comparison view. You can open Disassembly Comparison view by choosing **Windows-> Show View -> Other... -> Profile Analyzer -> Disassembly Comparison** or just find it in the right bottom panes.

To view the disassembly comparison, you should follow these steps:

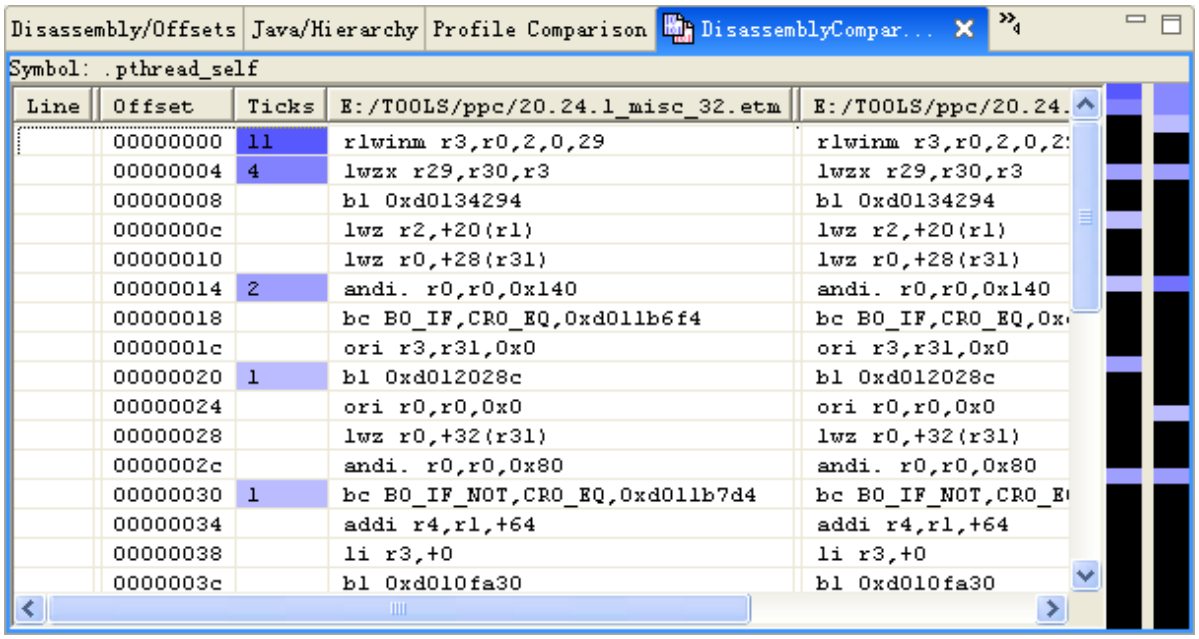
1. Compare two profiles through **Profile Comparison Wizard** or just open an .etc file.
2. Open Profile Comparison view by choosing **Windows-> Show View -> Other... -> Profile Analyzer -> Profile Comparison**



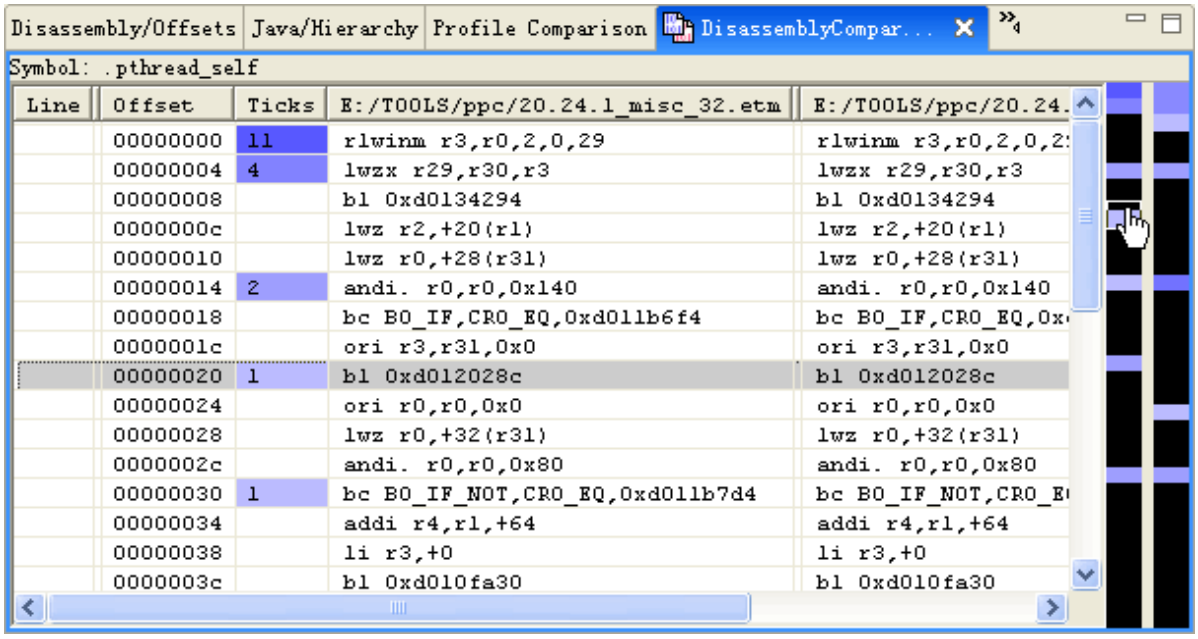
3. Right-click the line in Profile Comparison View and choose **Show disassembly comparison**



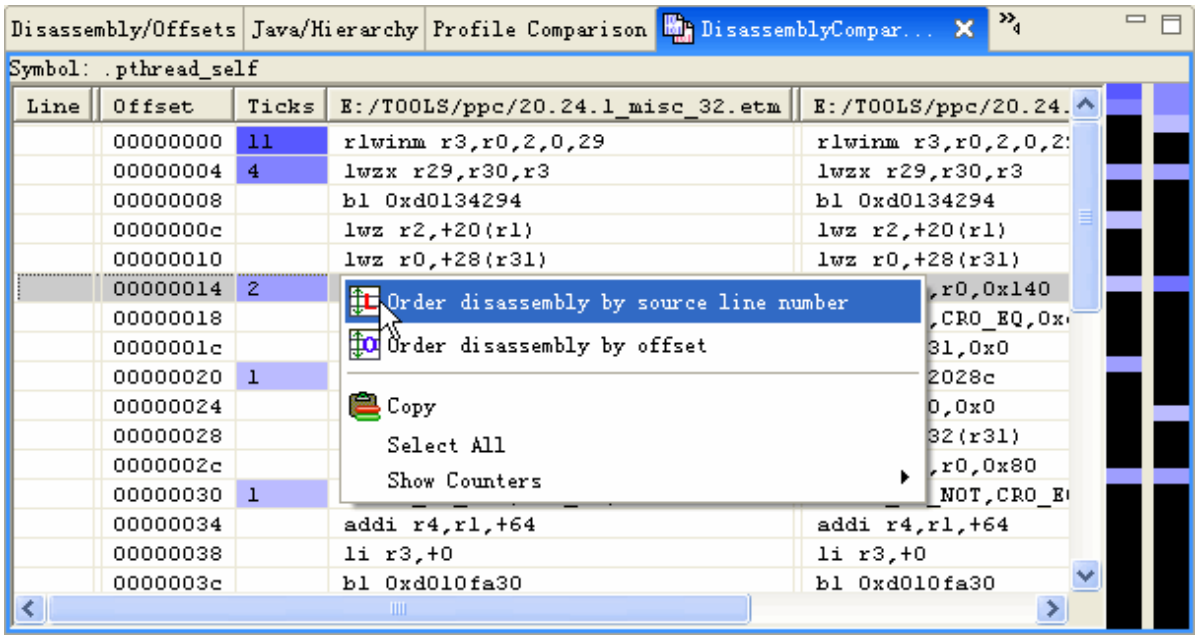
4. The corresponding disassembly comparison view opens.



There are two rows of hottest bars referring to two profiles which are compared. You can navigate disassembly comparison view by clicking them as follows:



To right-click and select menu item, you can sort columns by source line number or offset.



5.1.7.4View offsets and disassembly

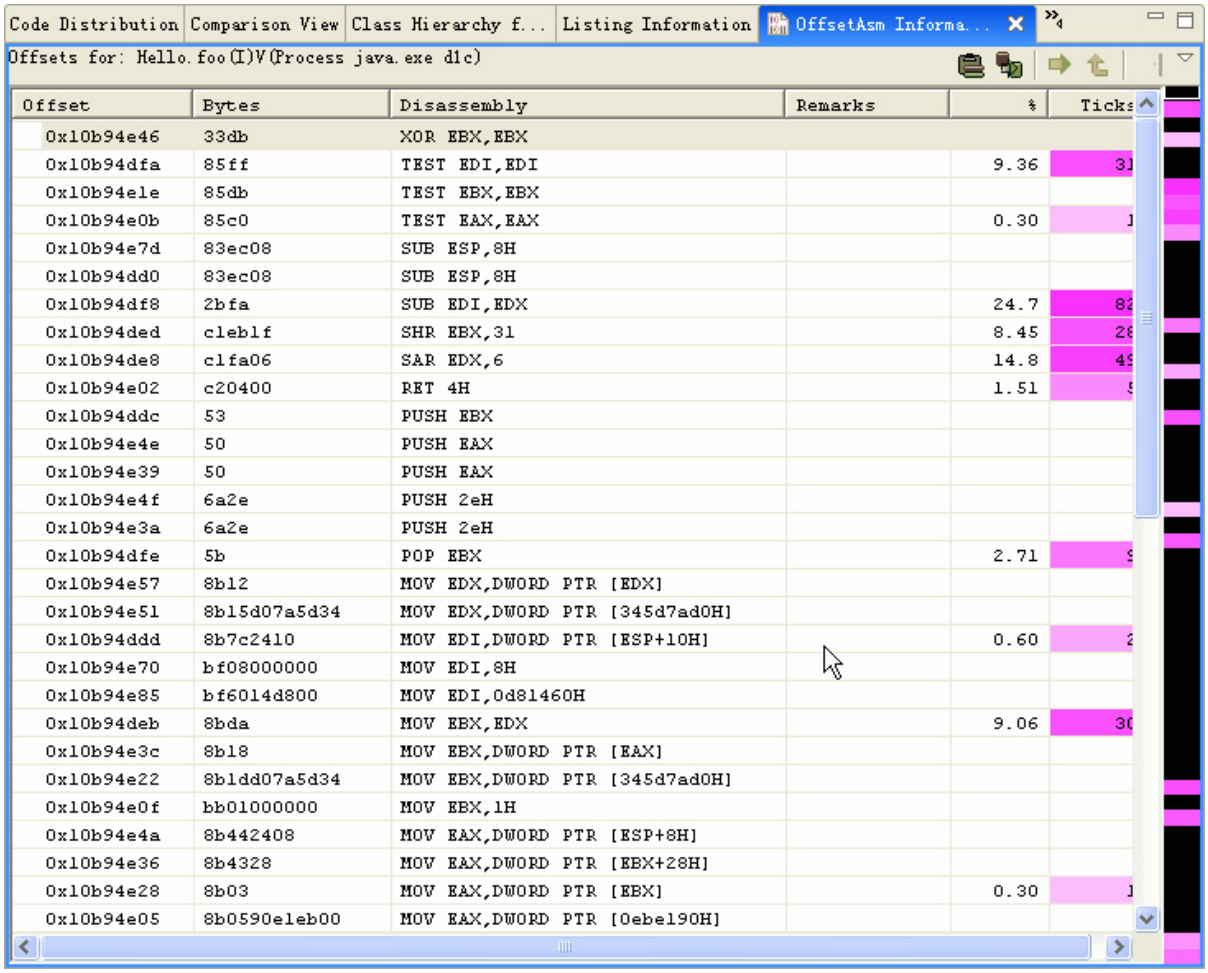
Profile Analyzer can disassemble the instruction stream for any symbol for which such a stream is available. Disassemble support is available for the following platforms:

- Intel IA32
- AMD-64 or EM64T (same instruction set)
- PowerPC
- zSeries
- CELL/B.E. (both PPE and SPE)

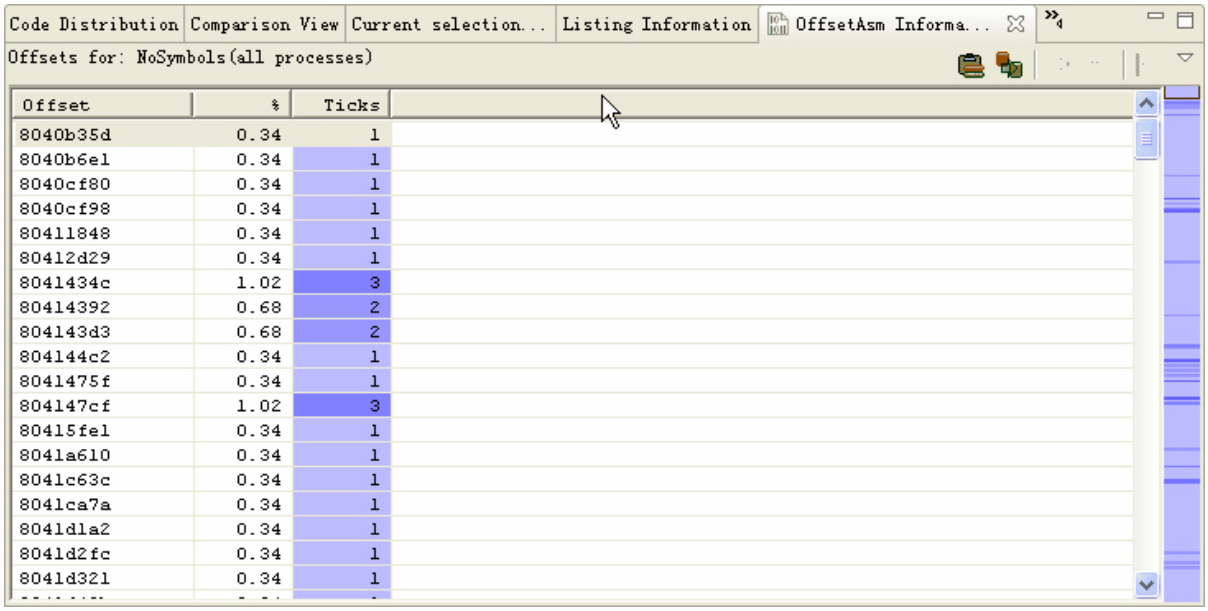
Whether the profile contains an instruction stream is dependent on the profiling tools used to create it.

For JITCODE (JIT-compiled Java methods), instruction streams are available if the JPROF library was loaded with the JVM (using the -Xrunjprof option), the jints sub-option was specified as part of this option, and the log-jita2n* files produced were available at the time that mergetprof was run. Only the IBM Virtual Machine for Java supports the JPROF library.

When disassembly can be generated for a symbol, Profile Analyzer displays a table containing instruction addresses, the bytes for each instruction, the instruction sequence, and tick information. The following view shows the disassembly for a Java method on an Intel IA32 system:



If no disassembly is available, Profile Analyzer displays an Offsets view containing ticks for each offset. The following view shows the offsets for the NTOSKRNL.EXE module of the same profile; this module has a single symbol referred as NoSymbols for that symbol data (and by extension, instruction stream of a symbol) could not be obtained currently:



If you are expecting to see disassembly data for a symbol and instead see only offset data, check the following:

There should be a tprof.micro section (for static-compiled methods) or a log-jita2n section (for JIT-compiled methods) in the profile you have loaded.

- The appropriate section should contain instruction data. In the tprof.micro section, the symbol must have a sequence of lines beginning with C:; if no such lines exist in the tprof.micro section, the -off option may not have been specified in the POST options (if you were manually profiling). In the JITA2N section, after each symbol there should be a sequence of binary bytes or hex data.

- You may be able to view the disassembly by switching to the Disassembly view. Click on the pulldown menu at the top right of the view and ensure that the **Show disassembly** item is checked.

5.1.7.4.1Navigating the Disassembly/Offsets View

You can quickly navigate to areas of high activity in this view using either the **Hotness bar** or a combination of sort and selection actions:

- **Navigate to hot areas by sorting and selecting**

  You can click on any column in the offsetAsm Information view to sort by that column. Repeated clicks on the same column reverse the previous sorting order. To navigate to hot areas in a symbol you can follow these steps:

  1. Click on a column heading that relates to CPU activity, to sort the view by that column (e.g. Ticks, %CPU activity, or a CPU counter if the profile contains CPU counter counts). The lines with the most events are sorted to the top.

  2. Select a line of interest; the top line should be the one with the most events in the column you selected.

  3. Click on the **Offset** column heading to sort by offsets again. The busy line you had selected in the previous step remains selected and remains in the viewable area.

  If your platform supports symbol call resolution (currently only the x86 and x86-64 platforms, as these are the only platforms in which direct relative or absolute branch instructions are used to make calls to other symbols), you can also quickly find calls to resolved targets by sorting by the **Remarks** column. The following shows disassembly for a JIT-compiled Java method, sorted by the Remarks column so that lines containing call targets are displayed at the top:

Offset	Bytes	Disassembly	Remarks
0x0256DED0	E8E9D0F5FF	CALL 24CAFBEH	java/lang/String.equals(Ljava/lang/Object;)Z_24cafb4
0x0256E920	E899C6F5FF	CALL 24CAFBEH	java/lang/String.equals(Ljava/lang/Object;)Z_24cafb4
0x0256F158	E861BEF5FF	CALL 24CAFBEH	java/lang/String.equals(Ljava/lang/Object;)Z_24cafb4
0x0256C7FA	E8FF5CF8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256DD41	E8B847F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256DDEA	E80F47F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256DFD8	E82145F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256E0D9	E82044F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256E7BB	E83E3DF8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256E85B	E89E3CF8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256EA2B	E8CE3AF8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256EB28	E8D139F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256EFC8	E83135F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256F06A	E88F34F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256F280	E87932F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256F378	E88131F8FF	CALL 24F24FEH	java/lang/StringBuffer.<init>(Ljava/lang/String;)V_24...
0x0256C808	E8D0A9FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256C832	E8A6A9FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256DD60	E87894FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256DE02	E8D693FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256DFE8	E8F091FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256E0F0	E8E890FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256E7D3	E8D58AFBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256E873	E86589FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256EA39	E89F87FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...
0x0256EB3A	E89E86FBFF	CALL 25271DDH	java/lang/StringBuffer.append(I)Ljava/lang/StringBuff...

You can use the same three-step sorting technique as for offsets, to find calls to a particular symbol:

4. Sort by the **Remarks** column. You may need to select the column header twice, if no call targets are visible the first time you select the column.

5. Select the line containing a call target of interest.

6. Sort by the offsets column. The line you had previously selected is now in the viewable area and instructions are displayed in offset order.

You can double-click on a line containing a call target to switch the current symbol in the offsetAsm Information view to the target symbol.

Loop nest detection and branch target detection

When Profile Analyzer loads the disassembly for a symbol it analyzes internal direct branches in the disassembly to determine loop patterns. Any backward branch may be considered the end of a loop, provided certain other parameters are met. Any block of code detected to be within a loop is indented by one space; if multiple nested loops are detected, sections of code may appear more deeply indented. In some cases the level of indentation may be extreme, as in the following example:

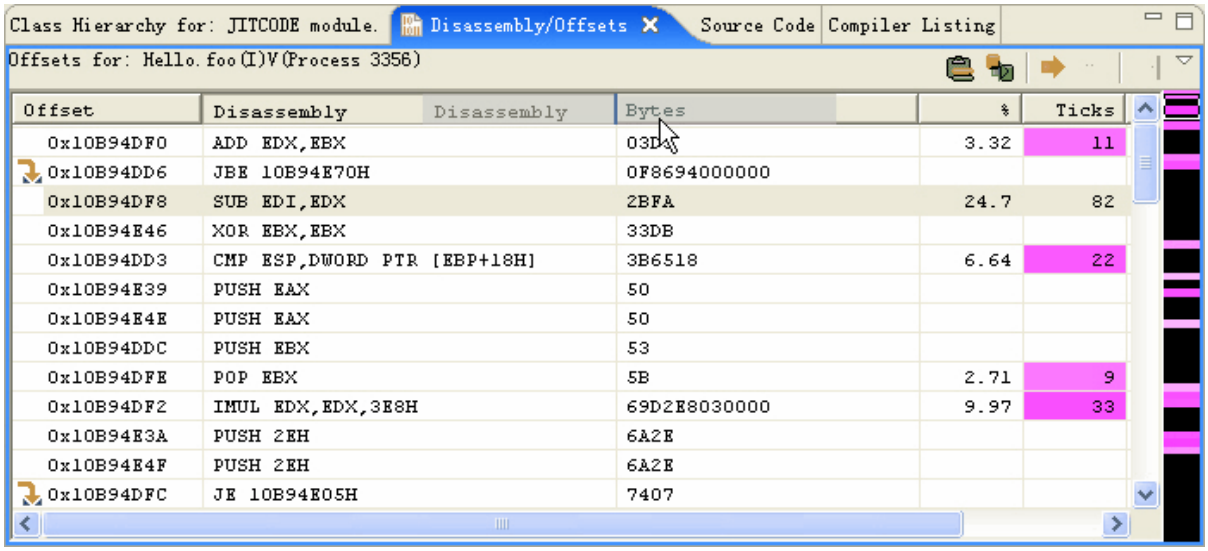
Offset	Bytes	Disassembly	Remarks
0x0256E819	48895128	MOV QWORD P...	
0x0256E81D	4833D2	XOR RDX,RDX	
0x0256E820	48895010	MOV QWORD P...	
0x0256E824	48895018	MOV QWORD P...	
0x0256E828	48895020	MOV QWORD P...	
0x0256E82C	4C8930	MOV QWORD P...	
0x0256E82F	488BD0	MOV RDX,RAX	
0x0256E832	48C1E20D	SHL RDX,13	
0x0256E836	4881E20000...	AND RDX,7FF...	
0x0256E83D	4881CA0E80...	OR RDX,800EH	
0x0256E844	48895008	MOV QWORD P...	

Here the indentation shows at least 15 levels of nesting. While it is unlikely that a programmer would have written a loop nest 15 layers deep, this level of nesting may occur where a compiler has inlined calls that occur within loops, and the inlined calls themselves contain other nested loops or further inlined calls.

You can remove loop nest indenting by clicking on the icon at the top of the view. If the icon is displayed, clicking on it will display loop nest indenting for a symbol whose disassembly was not indented.

Reordering columns

You can reorder the columns of the Disassembly/Offsets view to hide or show particular columns or change the order in which columns are displayed. This is one of the features of eclipse 3.1. You can reorder the column just by clicking on the column name and dragging it to the place where you want it to be. It shows as follows:



Offset	Disassembly	Disassembly	Bytes	%	Ticks
0x10B94DF0	ADD EDX,EBX		03D3	3.32	11
0x10B94DD6	JBE 10B94E70H		0F8694000000		
0x10B94DF8	SUB EDI,EDX		2BFA	24.7	82
0x10B94E46	XOR EBX,EBX		33DB		
0x10B94DD3	CMP ESP,DWORD PTR [EBP+18H]		3B6518	6.64	22
0x10B94E39	PUSH EAX		50		
0x10B94E4E	PUSH EAX		50		
0x10B94DDC	PUSH EBX		53		
0x10B94DFE	POP EBX		5B	2.71	9
0x10B94DF2	IMUL EDX,EDX,3E8H		69D2E8030000	9.97	33
0x10B94E3A	PUSH 2EH		6A2E		
0x10B94E4F	PUSH 2EH		6A2E		
0x10B94DFC	JE 10B94E05H		7407		

Branch and call navigation

Any disassembly line that contains a branch to a known target within the current symbol, or a call to another symbol, is indicated by an arrow in the left margin.

denotes a forward branch, one whose target is a subsequent instruction.

denotes a backward branch, one whose target is a previous instruction.

denotes a call or branch to another profiled symbol. This is only available for x86 and x86-64 platforms.

When you double-click on a line containing one of these icons, the view changes to show the target of the branch (a different location in the current symbol, or the target symbol of a call).

To navigate back to the last in-symbol branch you selected, after you have followed that branch, press the button on the offsetAsm Information view toolbar.

The hotness bar

By clicking on an area in the hotness bar, you will be taken to the corresponding disassembly instructions, or offsets. For lengthy disassembled methods, you may need to page up or down to find the hot area in question, as a line in the hotness bar that is one pixel high may relate to several pages of disassembly.

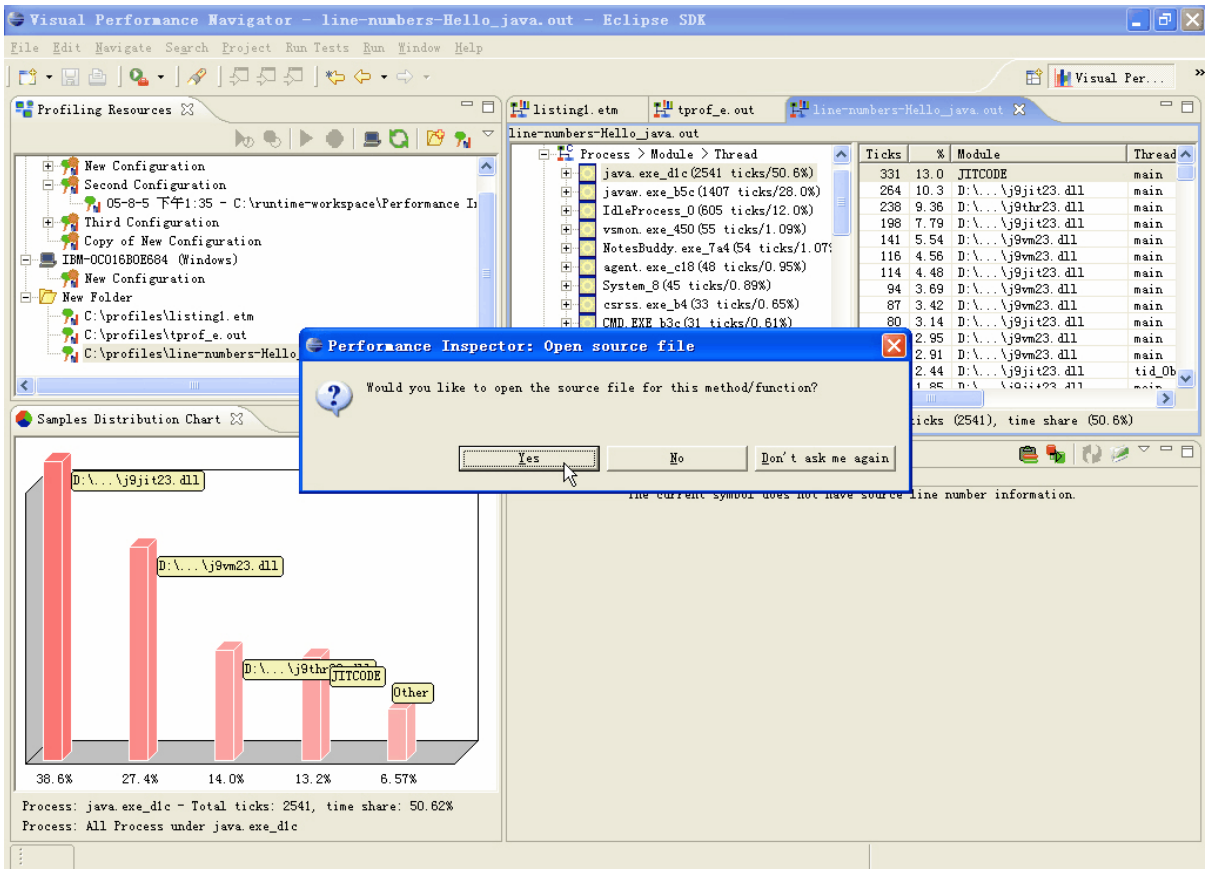
When you select a line in the Disassembly/Offset Information view, a yellow square appears in the hotness bar to show the currently selected area of the symbol.

5.1.7.5View source code

When an executable or library has been compiled with line number information (for example the -g option on some compilers), the platform profiler, like Tprof on AIX, may be able to obtain line number information for profiled symbols in such an executable or library. You can then view source code for these symbols within Profile Analyzer.

Line number support is available when the TPROF post-processing command includes the -off option. This option is enabled by default when you use the run.tprof_e script or run the profiling session from the **Profiling Configurations** view.

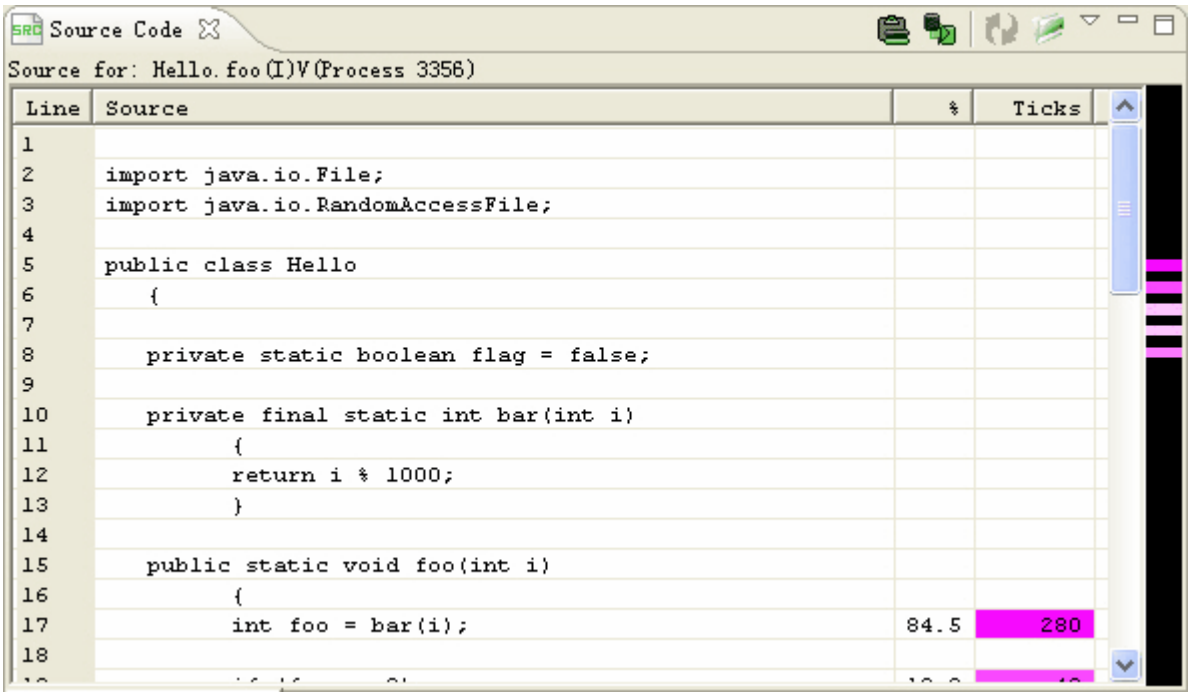
When you first select a symbol for which line numbers are available from the Symbols view, a dialog is displayed to ask whether you want to view source code for the symbol:



If you choose **Yes**, a File dialog is displayed that lets you navigate to the path containing the file. The name of the file you choose from this dialog does not have to match the name in the TPROF output, but if the line numbers do not match those of the file from which the code was compiled (for example, if the file has been edited since it was compiled), the tick information may not map to the correct source line numbers.

If you choose **No**, you are not prompted to enter source for any other symbols in the current profile, but when you load a different profile containing line number information, you may again be prompted to locate source files. If you choose **Don't ask me again**, you will not be asked to open a source file until you exit and restart Profile Analyzer.

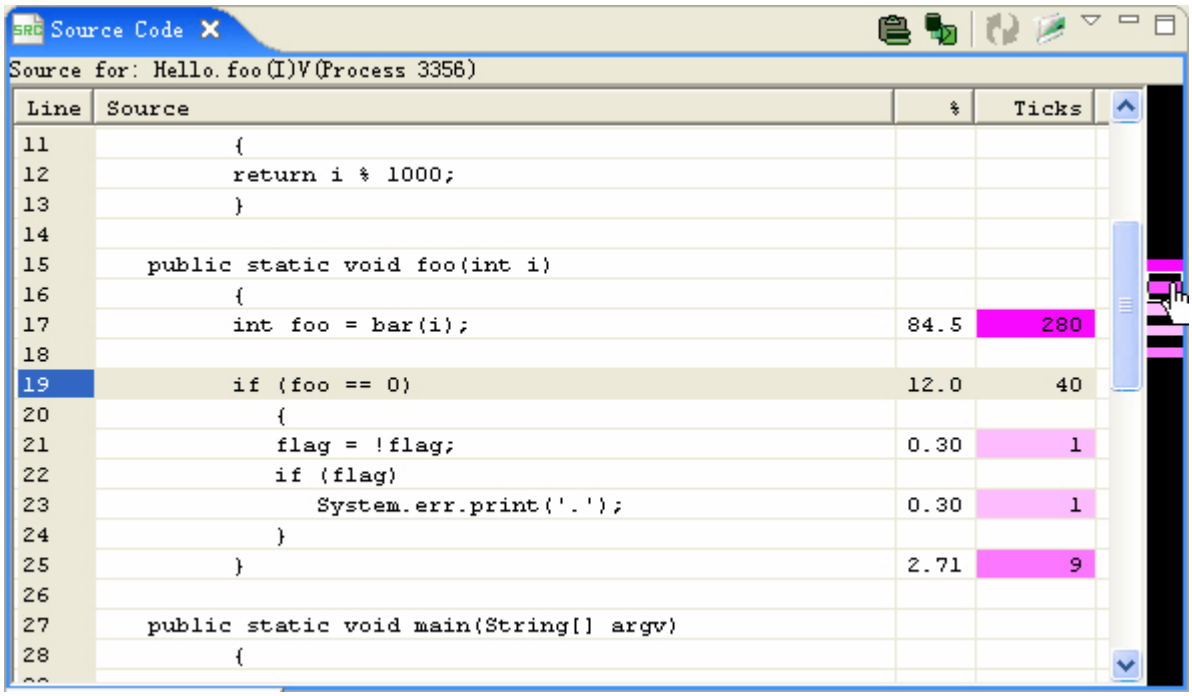
The following view shows source code for a symbol:



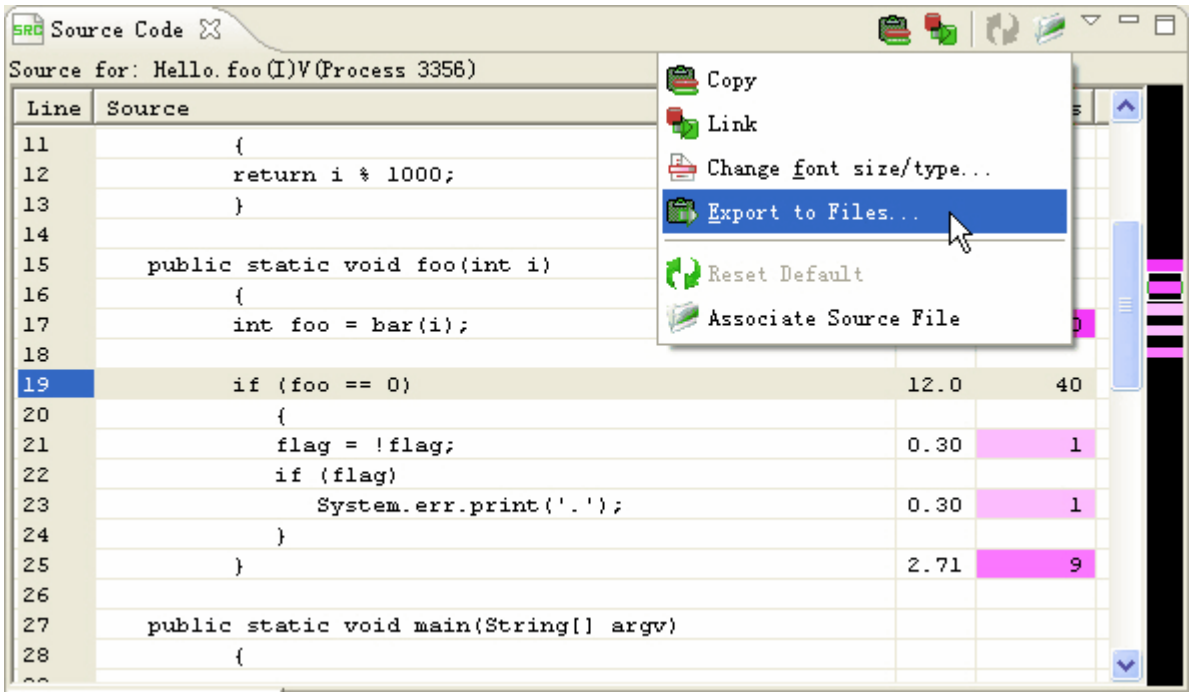
If you choose **No** or **Don't ask me again**, no source code will be shown. The source code view shows as follows:

You can again associate source file by pressing the button in the center of the view, or click **Associate Source File** icon on the toolbar of the view.

When you click on different areas of the hotness bar in the right side of the source code view, the corresponding line in the source file you select will be highlighted. You can see it in the following view:



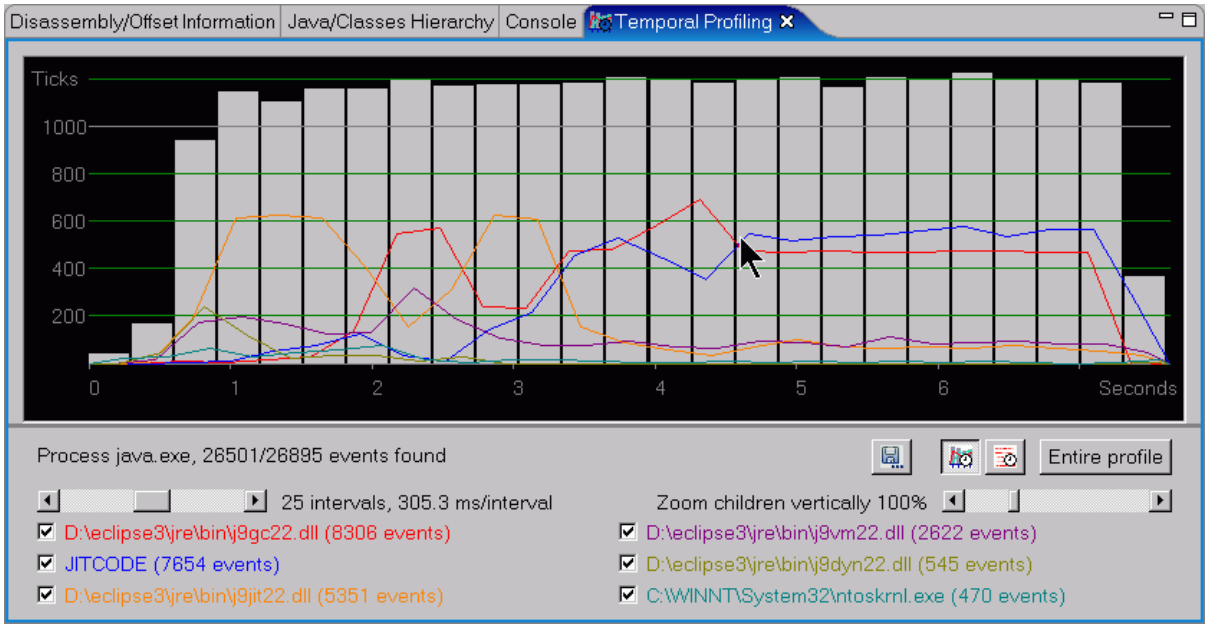
You can export these source codes to file by selecting **Export to Files...** in the menu of the toolbar as follows:



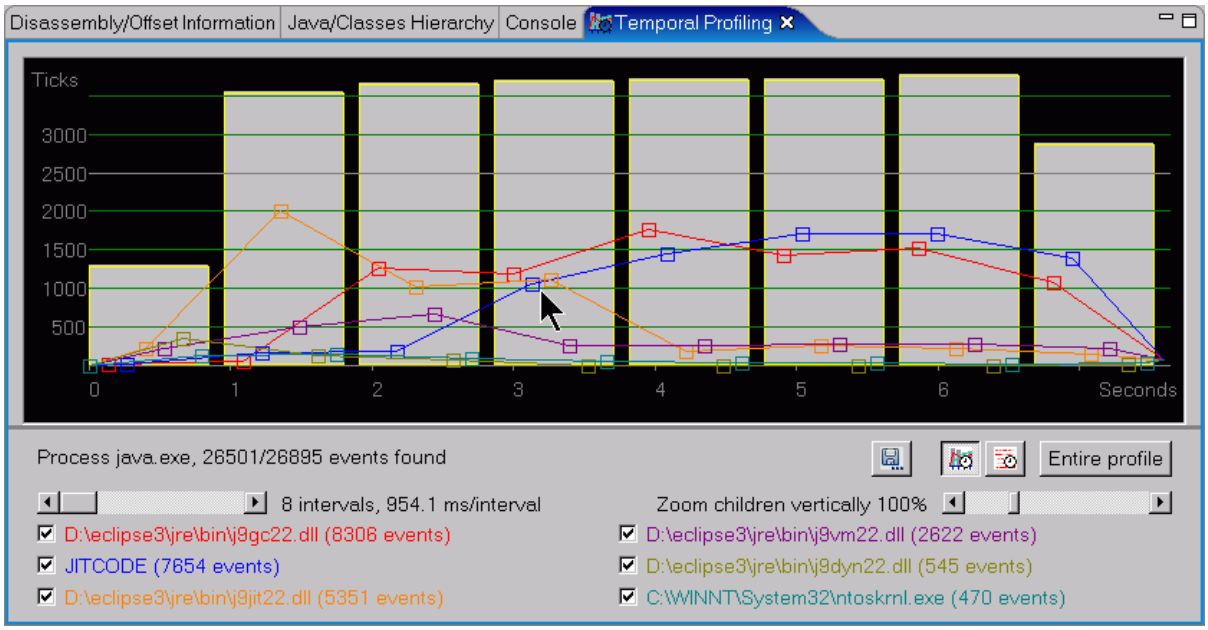
5.1.7.6View temporal profiling

When a Profile Analyzer profile contains a trace buffer section, Profile Analyzer attributes buffer events to appropriate symbol offsets, symbols, modules, threads, and processes. When you select a profile object (click on a process, thread, or module in the process tree, double-click on a symbol in the symbol list, or double-click on a disassembly or offset line with tick information in the offsetAsm Information view), Profile Analyzer can display a

temporal graph showing *when* during the profile run the ticks for that object occurred. This version of the Temporal profiling view is called **Tick intensity over time**. The following screen capture shows the Temporal Profiling view for a java.exe process in a profile that ran for about 7.5 seconds:

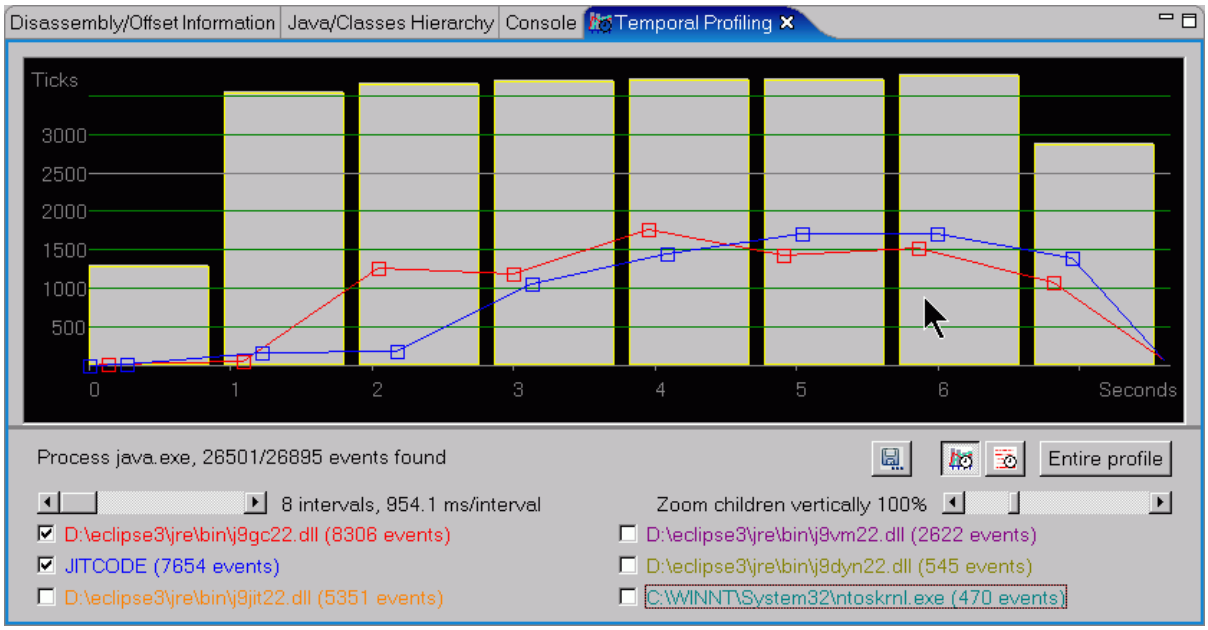


You can change the number of intervals of a temporal profile by sliding the intervals slider (on the left) to the left or right. This changes the number of intervals used to display the temporal graph. Changing the granularity of a 25-second trace run to 50 intervals will result in each bar showing the events for a particular half-second. For the same trace run, a granularity of 10 would have 2.5 seconds attributed to each bar. The following screen capture shows the same information as above, but with a granularity of 8 (that is, 8 equal-time intervals):



When the selected object is a Profile, Process, Thread, or Module, up to six of the "natural" children of that object are shown in a line graph superimposed on the bar graph, as can be seen in the above images. A child is only shown if it is a significant contributor to the parent tick count. You can hide the tick information for a child by

deselecting the check box beside its name below the bar graph. The following screen capture shows the same profile, with only the top two children selected:

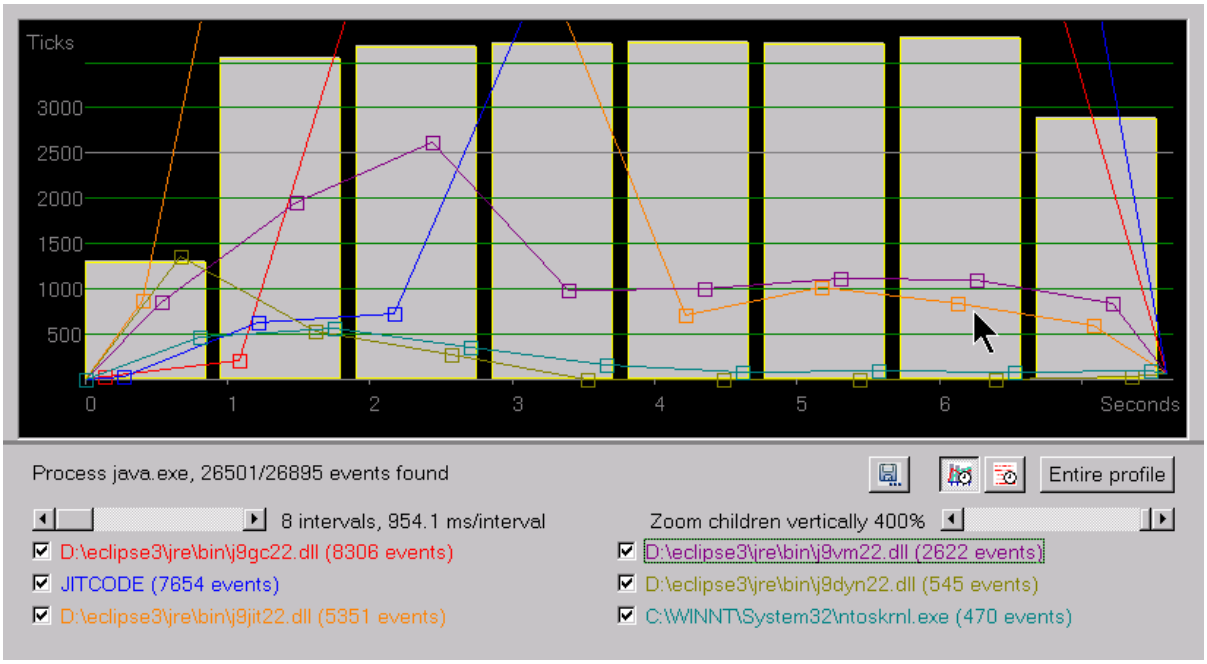


The "natural child" of a profile object is as follows:

Parent	Natural children
Profile	Processes
Process	• Modules, if "Ignore thread data" in the process tree checkbox is selected • Threads, if "Ignore thread data" checkbox is *not* selected
Thread	Modules
Module	Symbols
Symbol	No children
Offset tick	No children

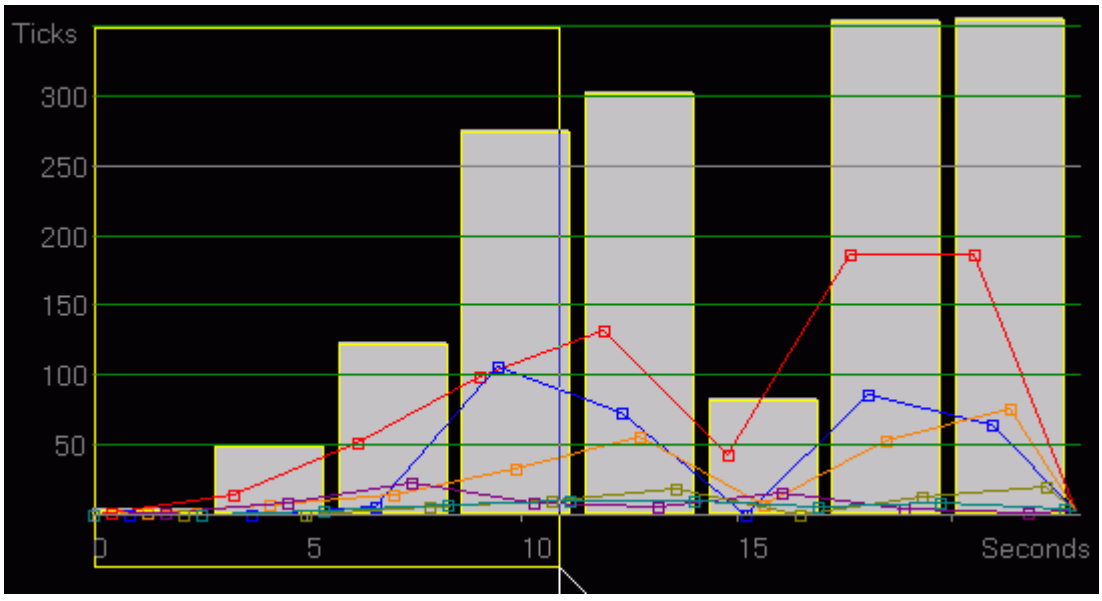
5.1.7.6.1Changing the tick scale for children

In some profiles, it may be hard to distinguish the lines of the "natural children" of a particular profile object. You can slide the **Zoom children vertically** slide bar to exaggerate or diminish the scale for the line graphs that are superimposed on the bar graph. The following shows the view immediately above, with all children selected and their vertical scale exaggerated by a factor to 400% of the parent object scale. The lines for the highest-contributing children extend off the top edge of the chart, but the difference in relative contribution of the three lesser modules is easier to distinguish because their lines are further apart.

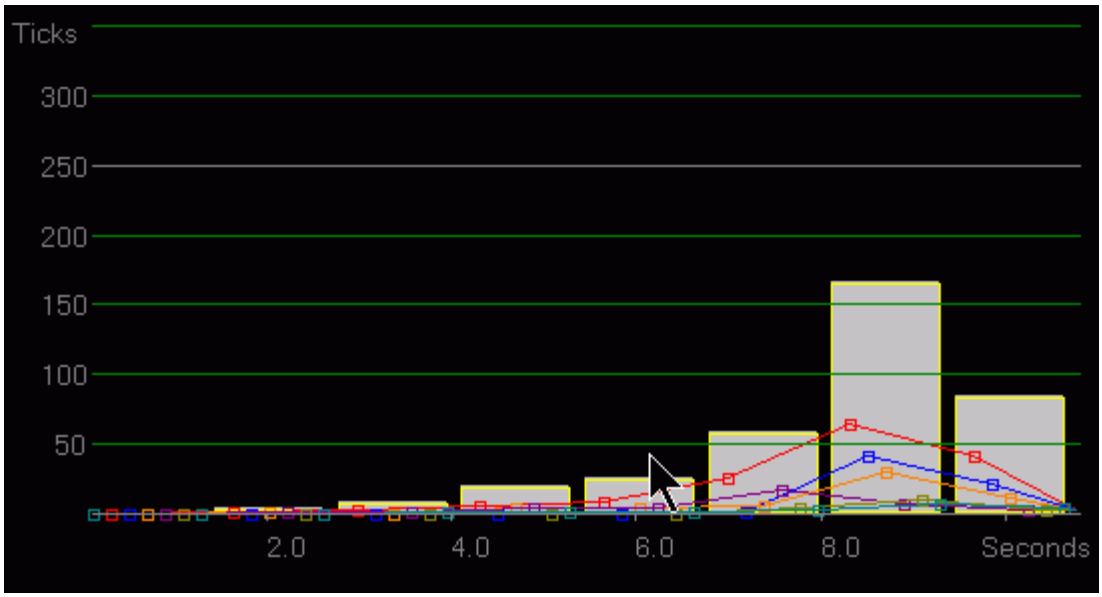


5.1.7.6.2Zooming in on a time range

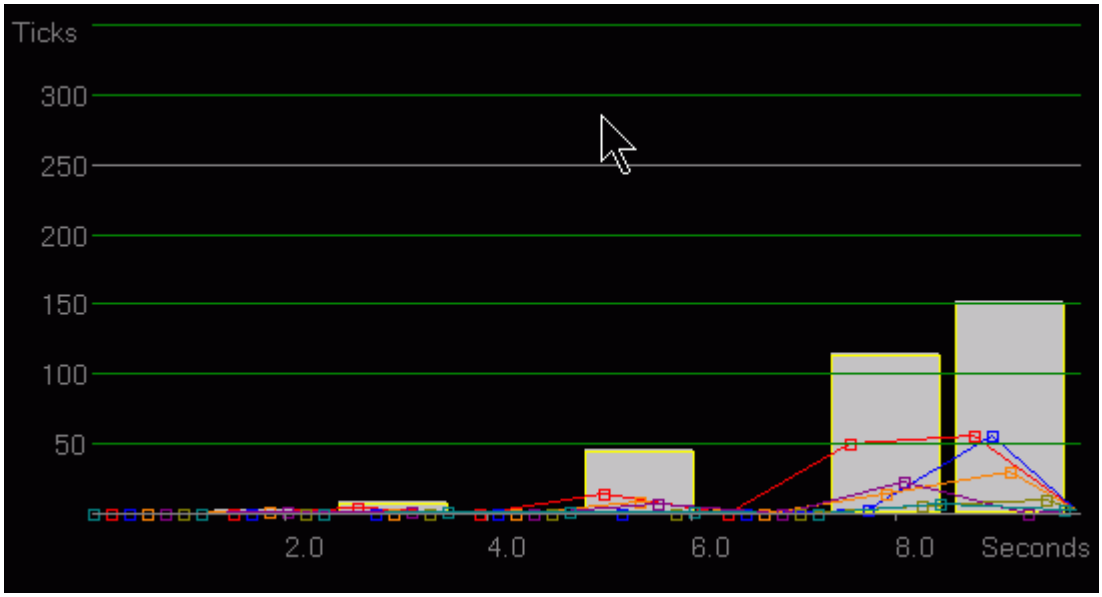
To zoom on a particular time range, press and hold mouse button 1 at one end of the time range, and drag left or right. A rectangle shows the time range selected (the vertical dimensions of the rectangle are not relevant to the result of the selection). The following shows the initial selection of a time range within a profile:



When you release mouse button 1 the selected area is zoomed:

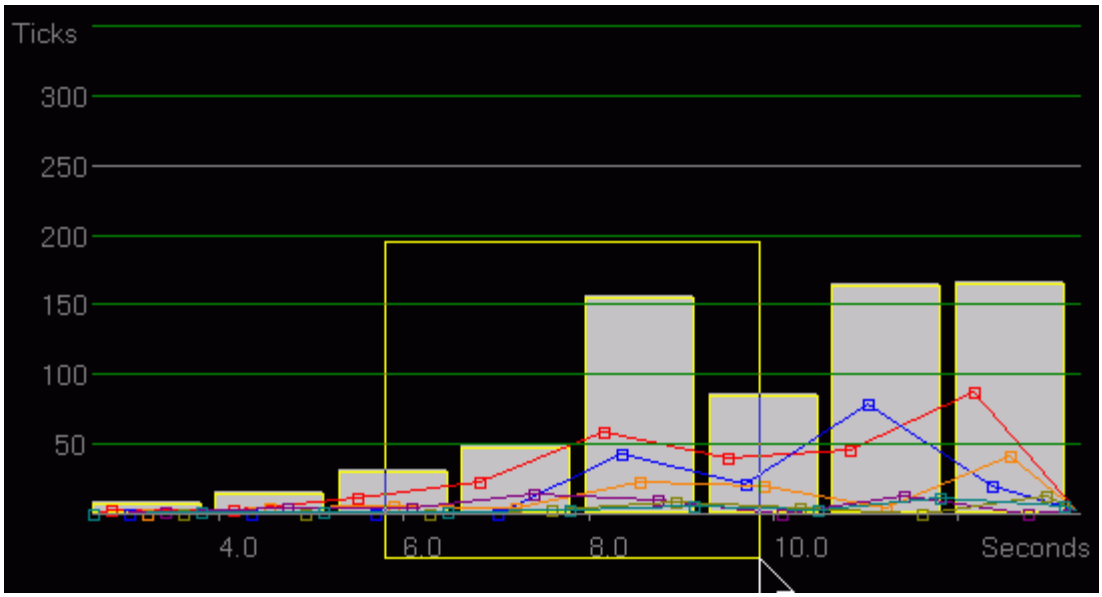


You can drag a zoomed view forward or backward in time by holding down mouse button 3 (the right mouse button for left-handed users) and dragging. The following animated graphic shows this effect:

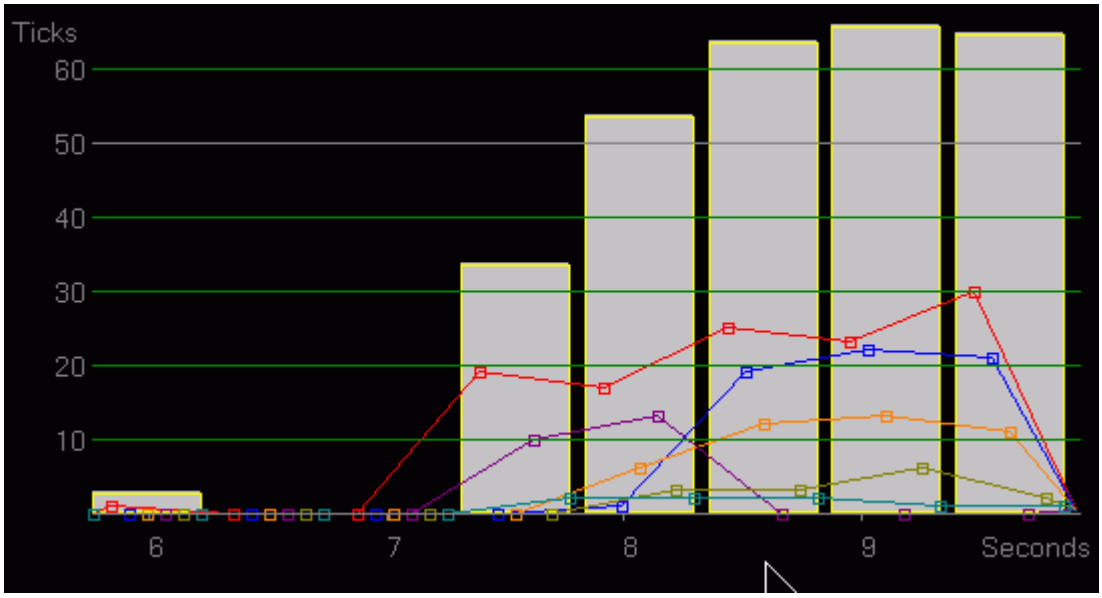


Note that the chart bars and lines move up and down as you drag: this is because, each time Profile Analyzer handles an increment of the drag event, it redistributes the trace events acording to the current co-ordinates.

You can further zoom an already zoomed image by selecting a new zoom area:



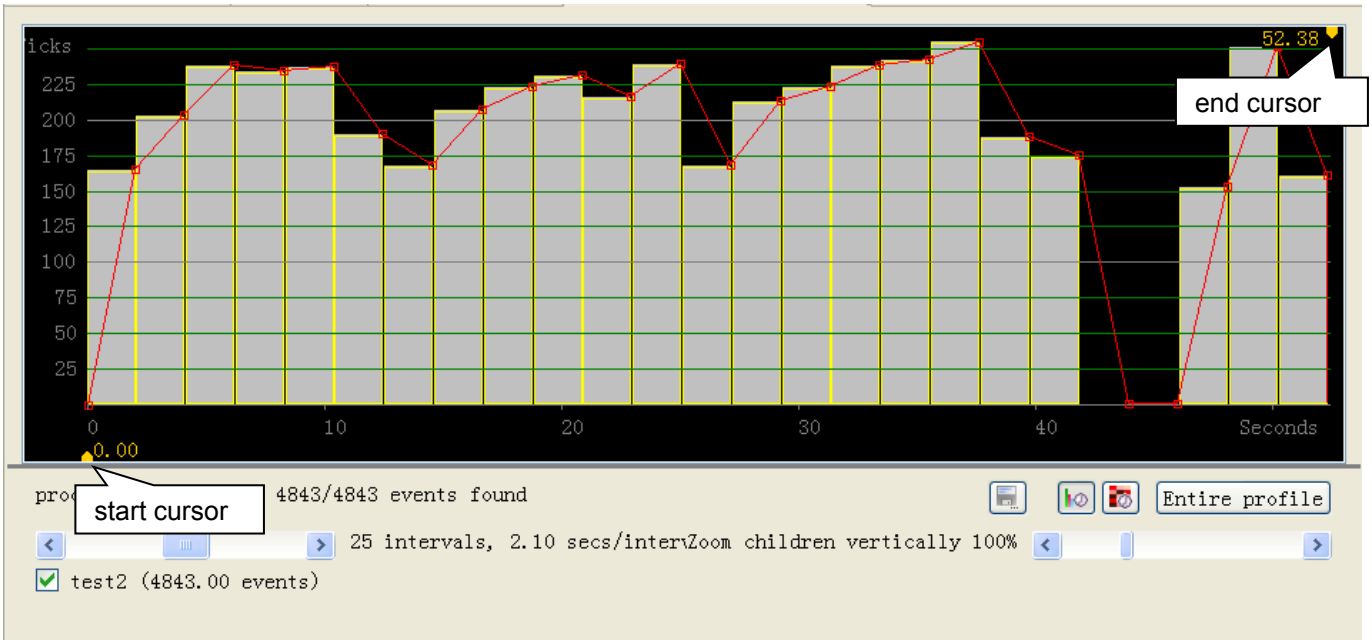
This zooms to the time range 5.76 to 9.10 seconds:



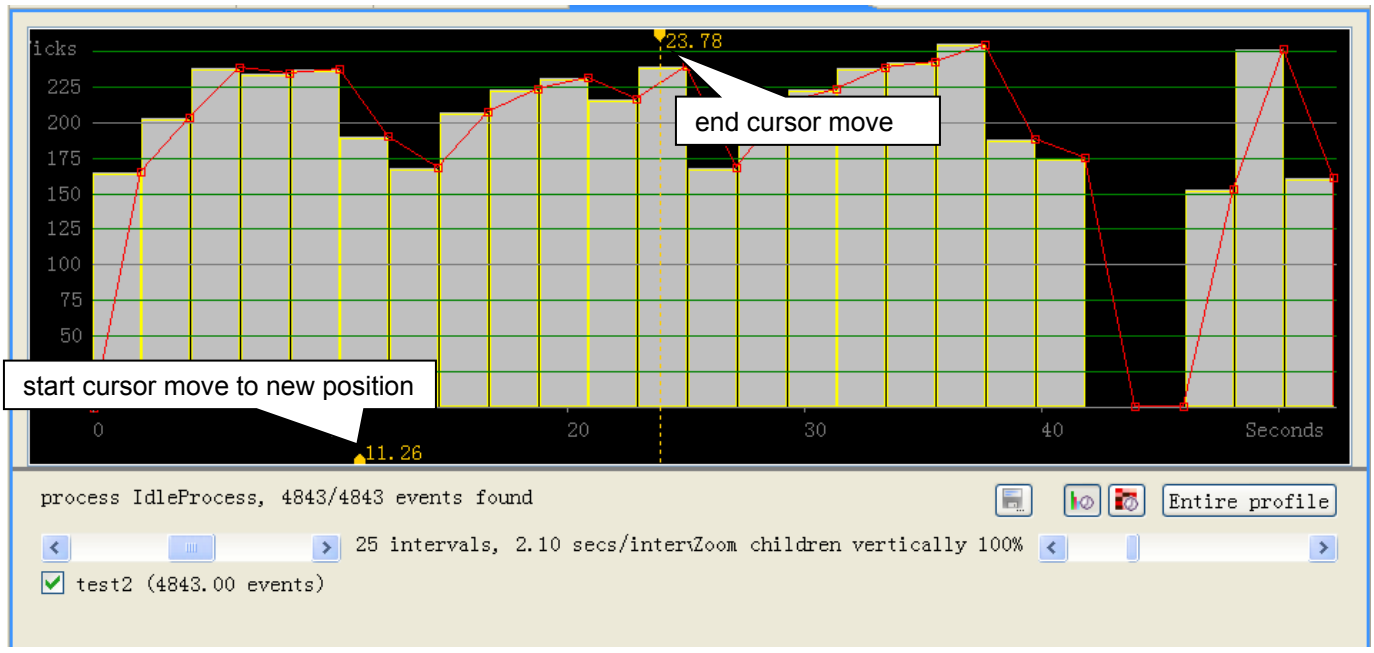
To restore the time view to the full duration of the profile, right-click over the bar chart.

5.1.7.6.3Time cursor

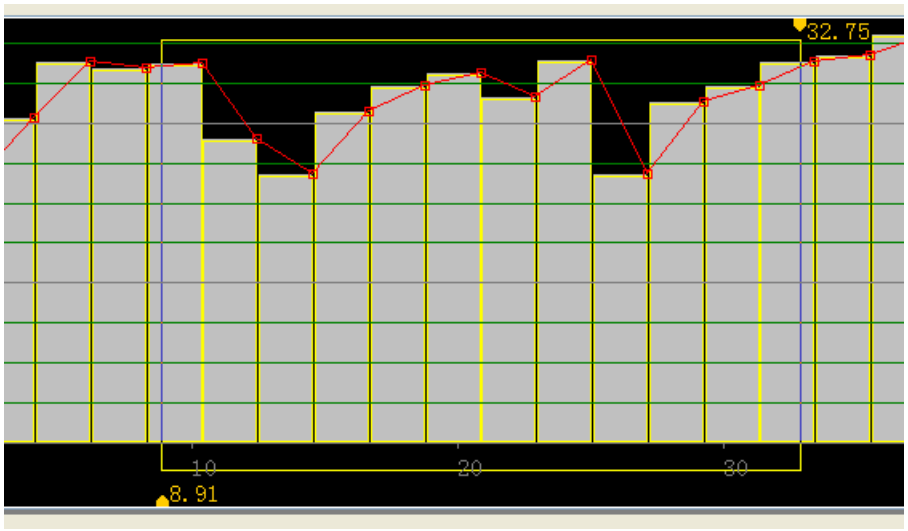
From VPA 5.0 on, temporal profiling view is decorated with two time cursors, one for start time, and the other for end time. Temporal profiling view displays a column chart of time intervals of all the profile data. Time is labeled along below the column chart, once every some time intervals. The time cursor helps user accurately position the time line of the column chart. When user selects the cursor and drags it along the column line, cursor time is displayed as it is moved. At first, the start cursor is at the bottom left and the end cursor is at the top right.



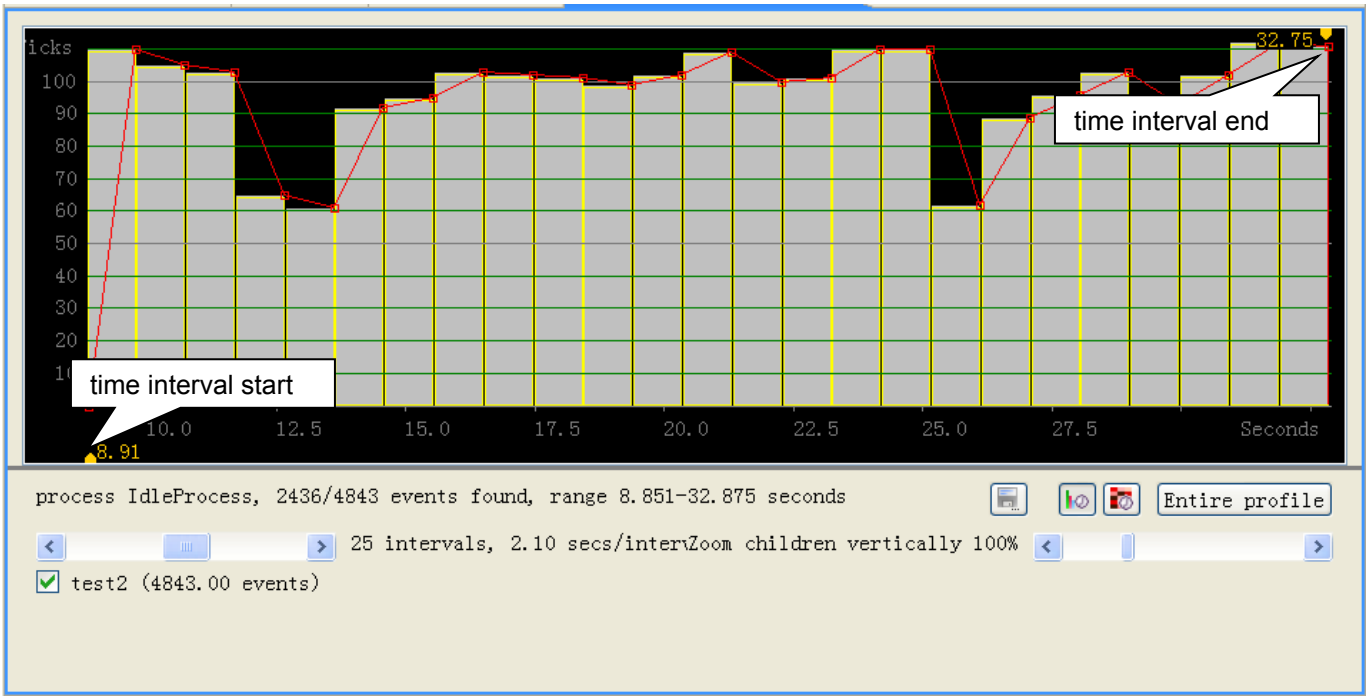
it is the temporal profiling view for a .etm profile data file with time data. The two cursors are displayed at the initial positions in the column chart.



When user drag the cursors , time line is displayed and time data is changed as the cursor is moved. Here the start cursor is labeled 11.26, and the end cursor is dragged to 23.78, in second.



Time cursor helps user focus on a range within a time interval, and helps user position the mouse to select a concerned range of time interval. In the above graph, the cursors having displayed a range of time interval, if user is interested in the interval of profile data, he can drag a rectangle aligned with the cursors. It helps user ranged area of time intervals more accurately.

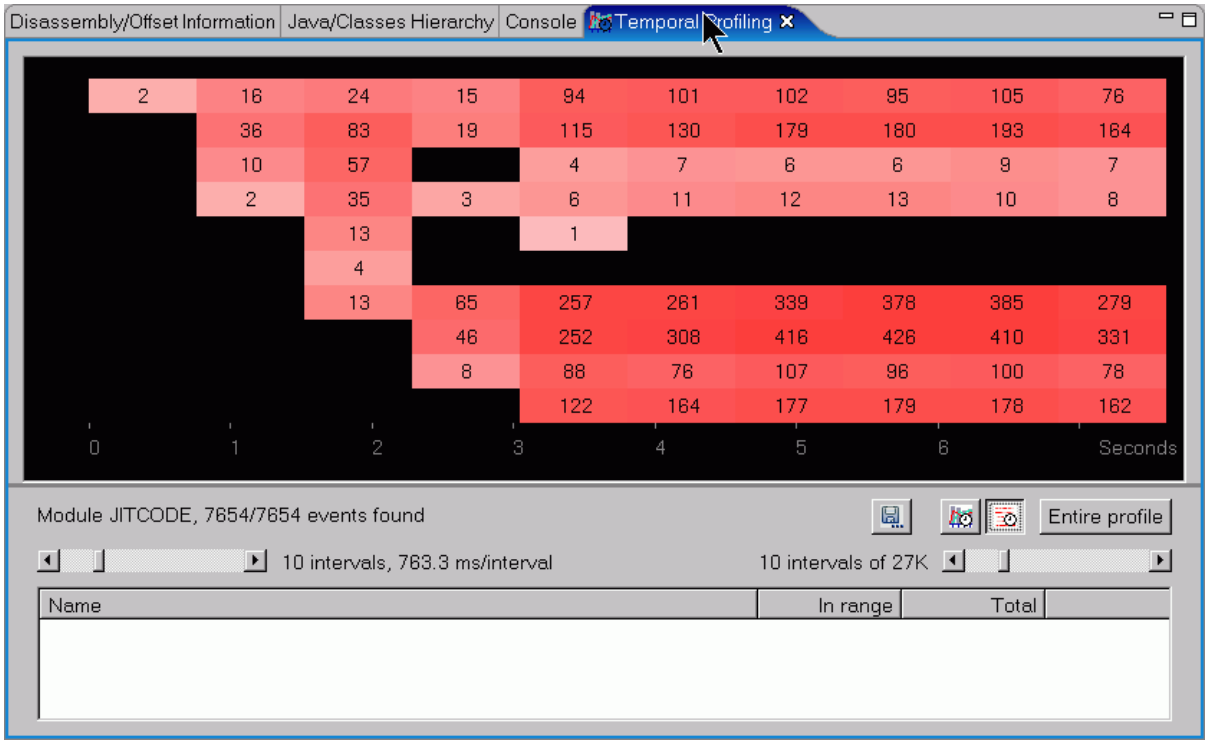


When user selects a time interval range with mouse, temporal profiling view refreshes the graph, and displays new column chart of profile data inside selected time interval. The above chart is resulted from selecting time intervals range from 8.91 second and 32.75 second. The start cursor is at the left most and the end cursor is at the right most. In this graph time cursor is able to move its position too.

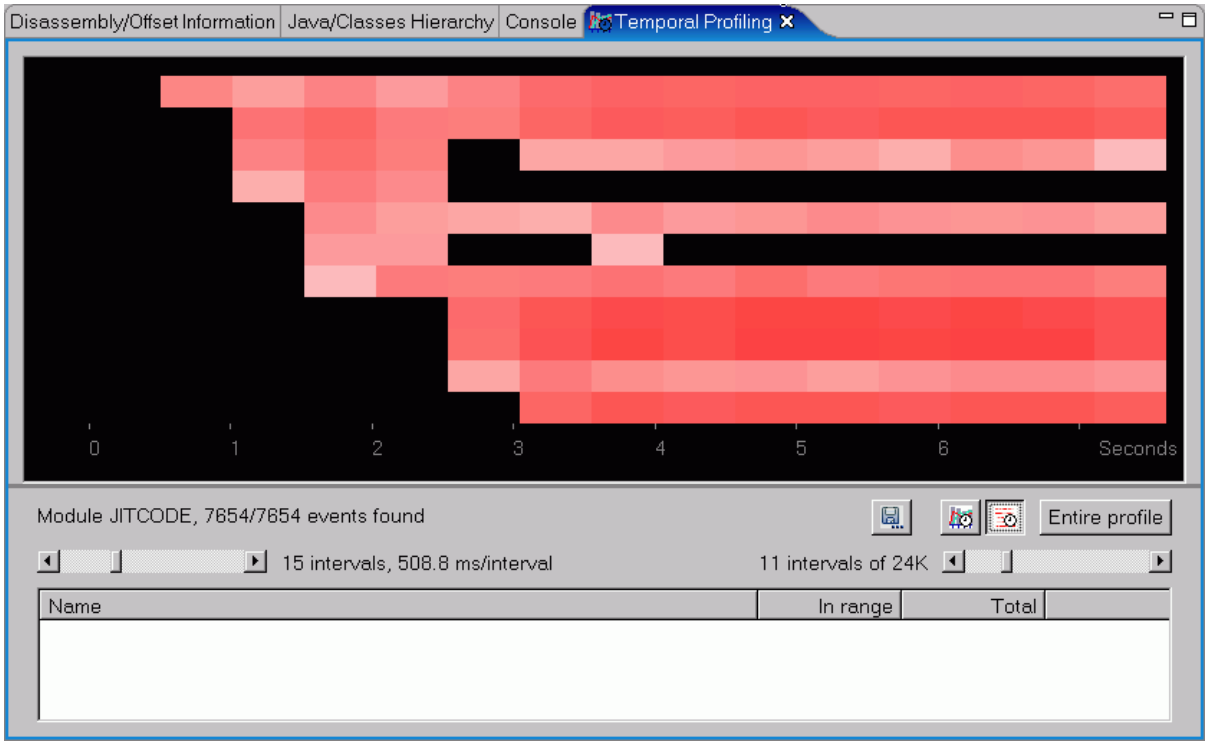
5.1.7.6.4Time/memory profiling

The temporal profiling view also lets you view how tick events for a selected object are distributed within a matrix of memory and time. This is mainly of interest to compiler writers, or other specialists concerned with how well busy sections of a symbol or module are distributed within a processor's instruction cache. To switch to the

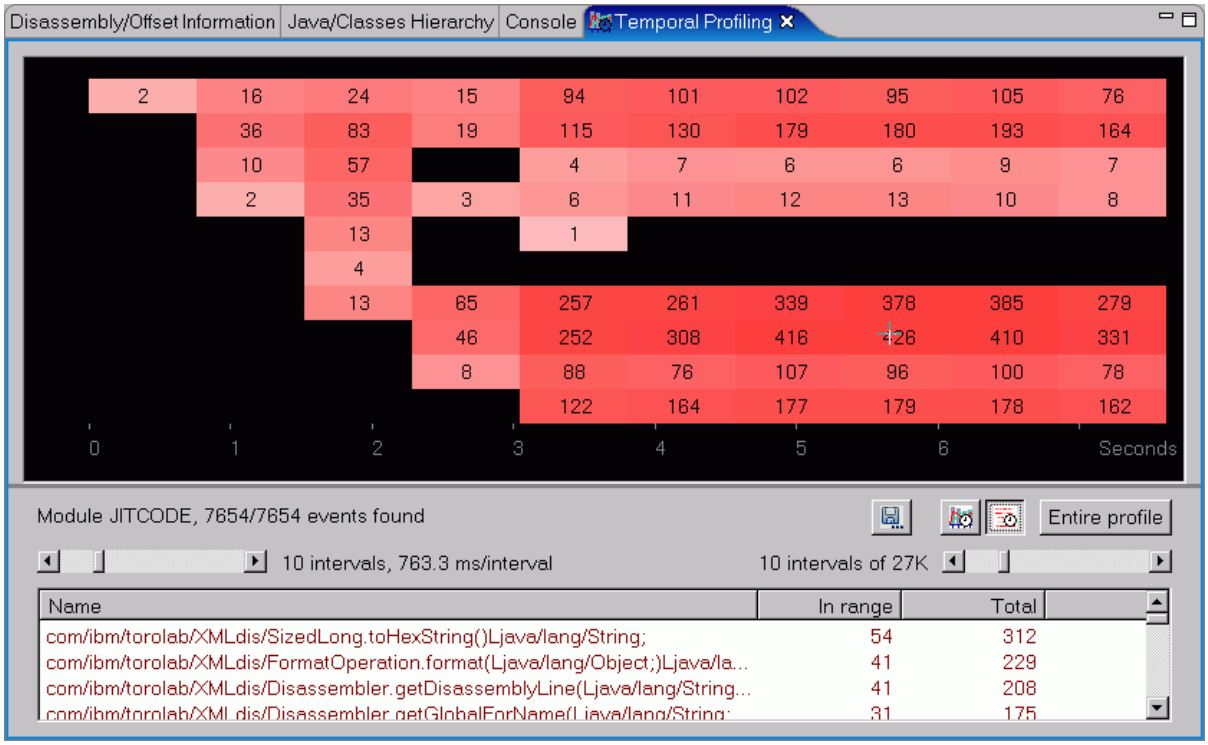
Time/memory profiling view select the **Show memory usage over time** icon . Select a profile object for which this view makes sense - typically a symbol or module. (It is not normally productive to view the entire profile or a particular process in this view, because individual libraries within the profile or even a single process may occupy widely different memory ranges.) The following shows a time/memory view of the JITCODE module of a java process (the module containing JIT-compiled Java methods):



Individual colored rectangles in this view represent profile intensity, for a given time and a given address range. To produce this view, Profile Analyzer divides the profile ticks for the selected object (profile, process, module or symbol) into equal time intervals (determined by the left-hand or time interval slider, as for the **Tick intensity over time** view) and divides the memory range of the selected object into equal memory intervals (determined by the right-hand or memory interval slider). Individual rectangles that contain ticks represent a region of memory that was busy at a particular time. If there is sufficient space, Profile Analyzer displays the tick count for each active intersection within the rectangle. If you increase the number of intervals the ticks may disappear but the color scheme still gives an indication of which memory ranges are busy at what times, with darker shades denoting higher tick counts:



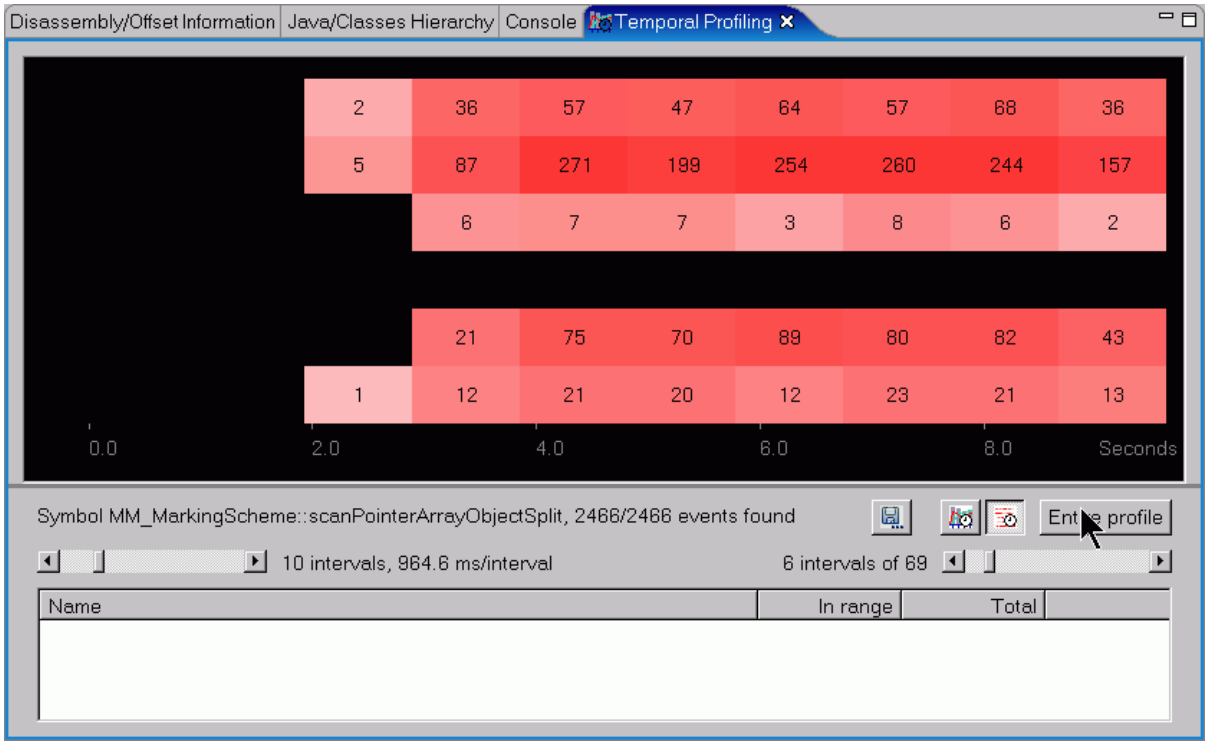
The table at the bottom of the view displays information about the active objects in a particular rectangle in the view. To use this capability, make sure the Temporal Profiling view is the active view (click on the tab), then **hold down the Shift key** while moving the mouse. As the pointer moves over different areas of the graph the objects that occupy the memory range for the current area are shown in the table. Two tick counts are shown for each object: **In range** identifies the number of ticks the indicated object contributes to the current time/space interval, while **Total** represents the number of ticks the indicated object contributes to the profile as a whole. In the following view, the Shift key is being held down and the pointer is over the rectangle with a tick count of 426. The busiest symbols for that time/memory range are displayed at the top of the table:



Once you release the Shift key, the table contents do not change. This allows you to use the mouse to scroll through the table after you have chosen a particular rectangle, without mouse movements changing what is displayed in the table.

5.1.7.6.5How to use the time/memory view

The time/memory view below is for a static compiled function within the garbage collection module of the J9 virtual machine for Java:



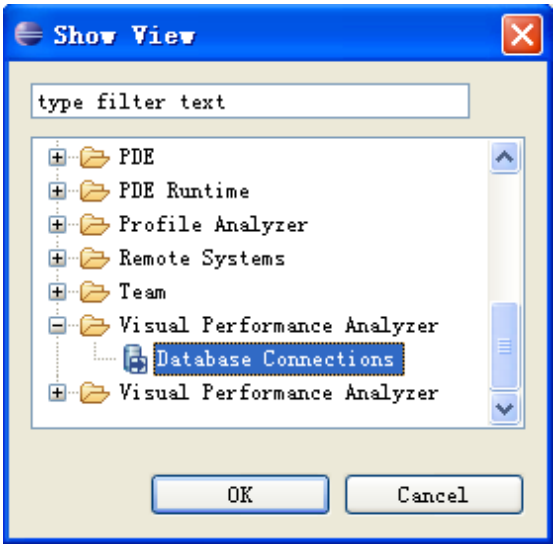
In this view, each row represents 69 bytes (as shown by the legend beside the right-hand slider bar). You can see that there are two fairly busy ranges: the first range consisting of the top two rows, and the second range consisting of the fifth row (the row whose first displayed value is 21 ticks).

One use of the time/memory view is to show whether code is properly ordered within busy symbols. For instance, the above function might provide better performance if the areas of code that are busy were closer together in memory, as they would likely use fewer I-cache lines if grouped together. Note that you should only attempt code reordering based on the time/memory view after analyzing the same symbol in several profiles, and there are no guarantees that your reordering will yield improvements. For example, compilers may completely reorder sections of your code when they generate the machine code for a symbol. However, this view may help you identify symbols or modules with time/memory usage patterns that warrant further investigation.

5.1.8Configure database connections and manage cached database files

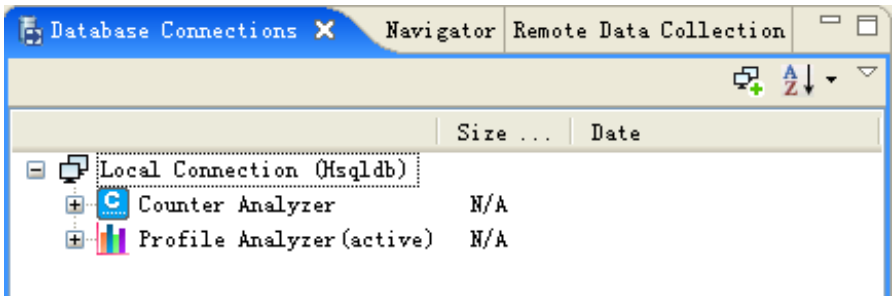
To improve the performance, Profile Analyzer will load its files into a database, either hsqldb or DB2, instead of keeping them in the memory, sometimes in page files. Then, each action just executes a query to get the data needed without any useless data.

To open this view, click Window → Show View → Other, then, select Database Connections under Visual Performance Analyzer category, as below.



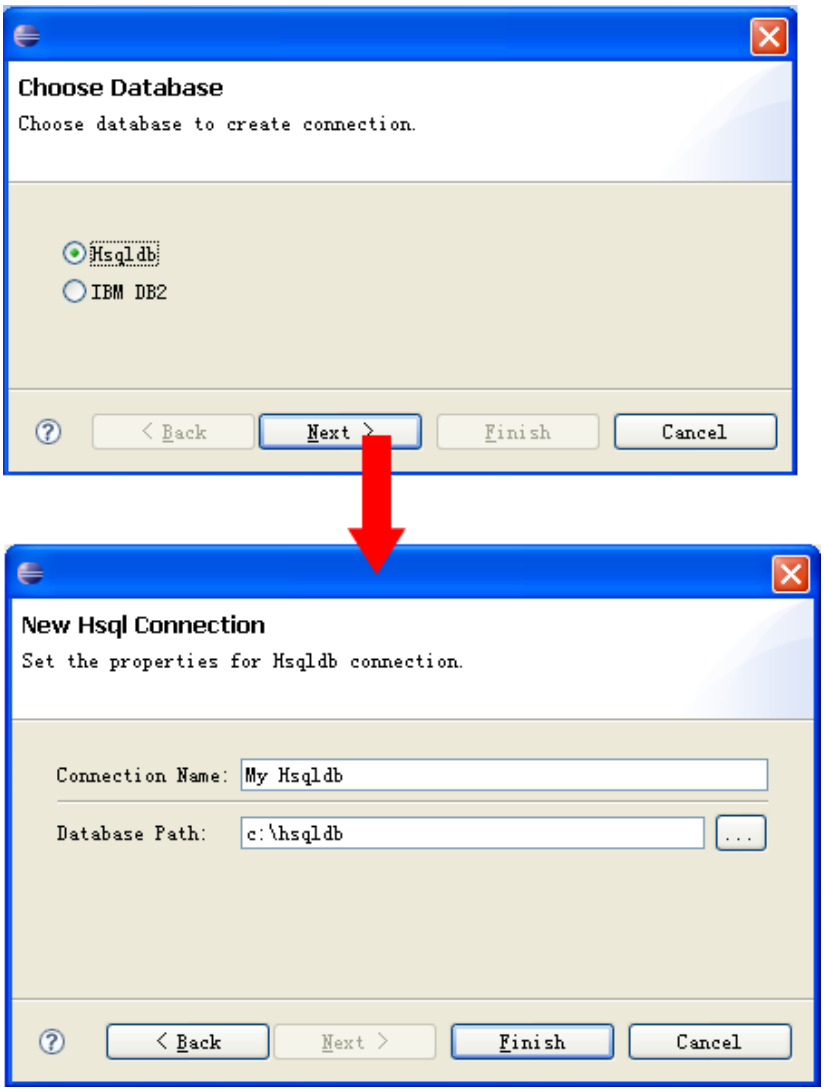
Open View

When you start VPA for the first time, a default connection of hsqldb will be created, as well as product supports under this connection node, as below.

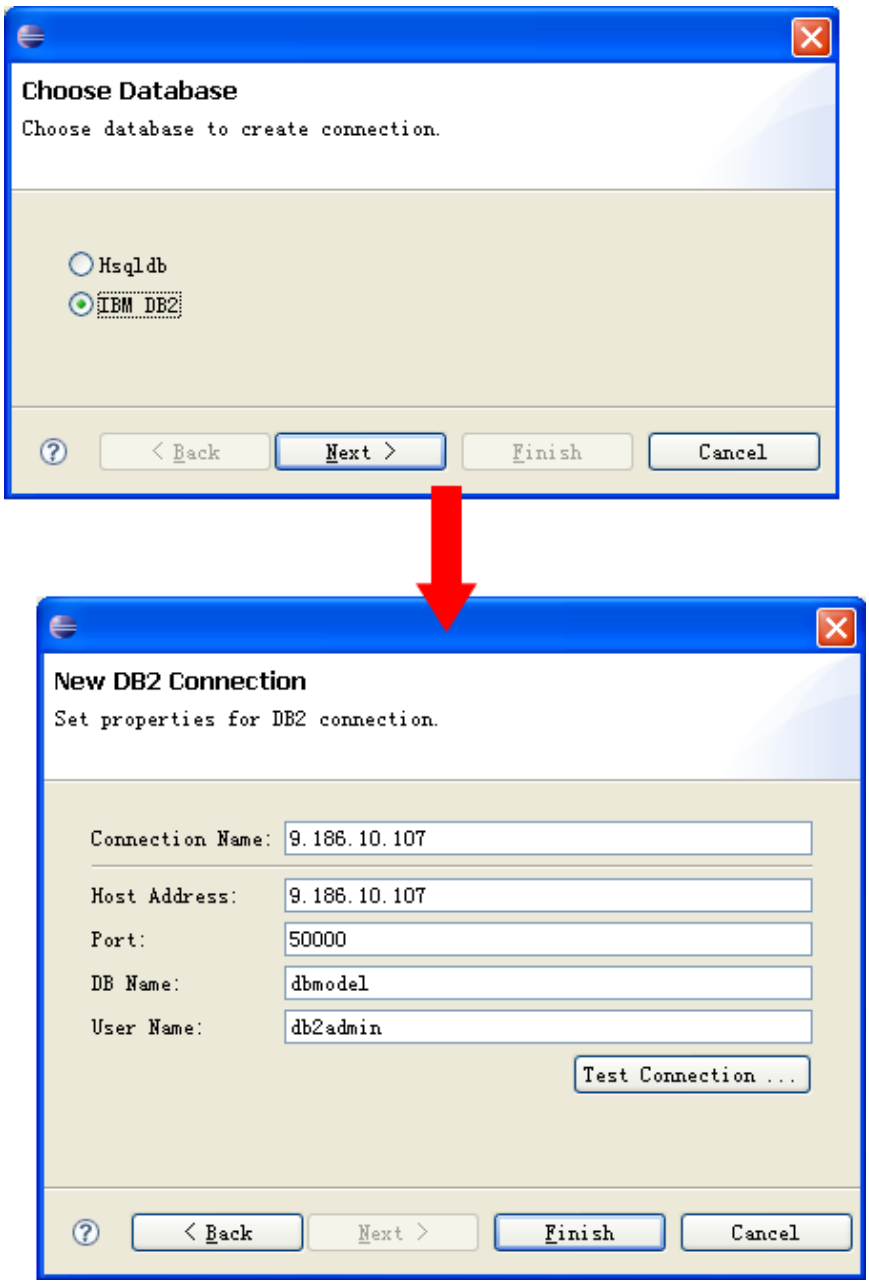


Default Connection and Product Supports

The only Profile Analyzer support is set as active, so that user can open files without any setting actions. However, you can create other connection or edit this default connection as you like, such as modifying its path.

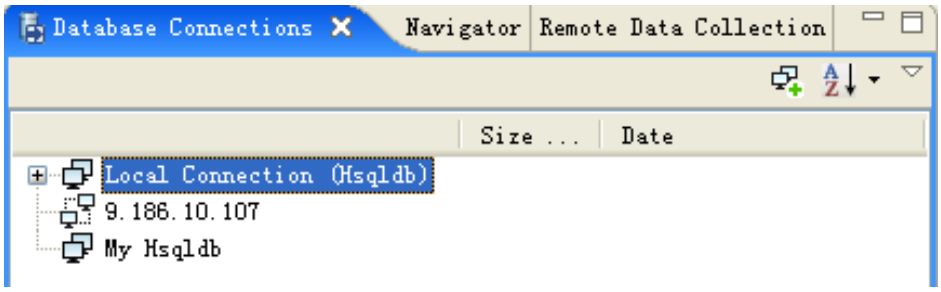


Create Hsqldb Connection

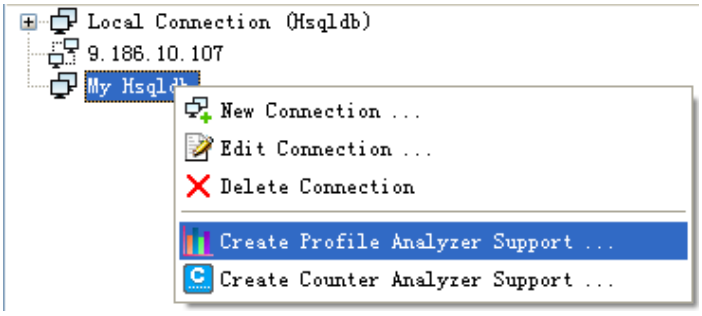


Create DB2 Connection

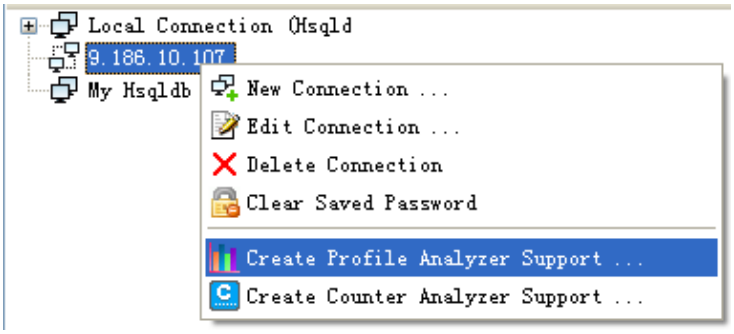
After these two connections have been created, the view looks as below.



User can create Profile Analyzer support under each connection, for common use, hsqldb is enough, for performance consideration, db2 is the better choice.

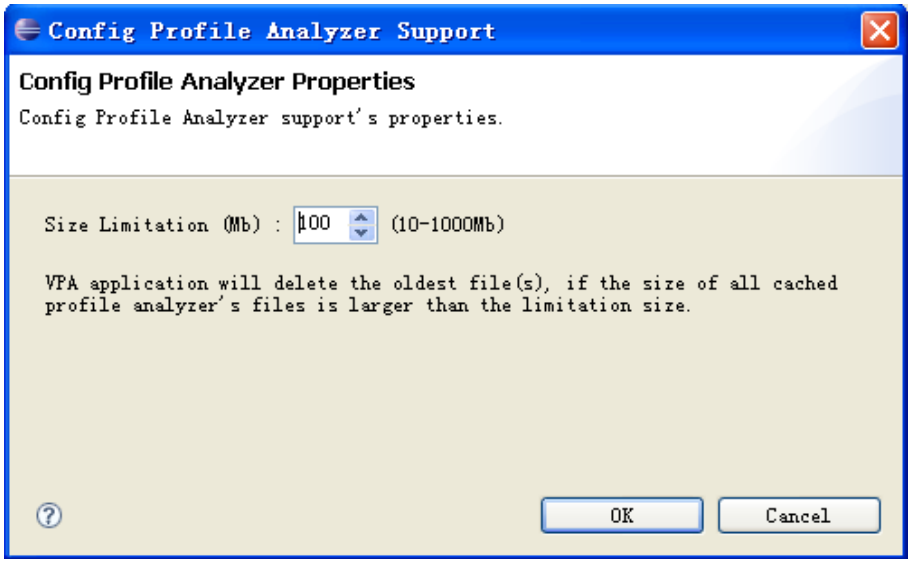


Create Profile Analyzer Support under Hsqldb Connection



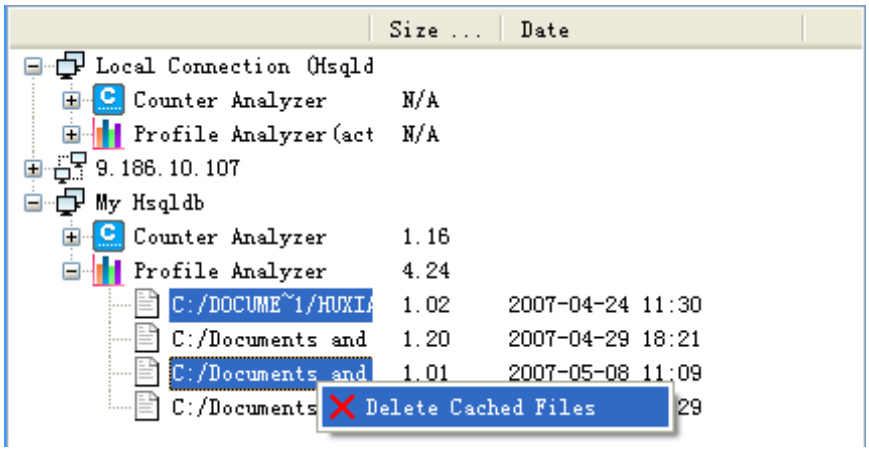
Create Profile Analyzer Support under DB2 Connection

Hsqldb, as we known, is an embedded database system, so it must have some limitations. To prevent too much disk space occupied as more and more files being loaded in, you should set a size limitation. This num of Size Limitation stands for the upper limit of the hsqldb's data file. If the database file was larger than the num, system will delete oldest files, until the size of the database file is smaller than the size limitation. During the auto delete process, some file whose size is larger than the size limitation will be deleted first.

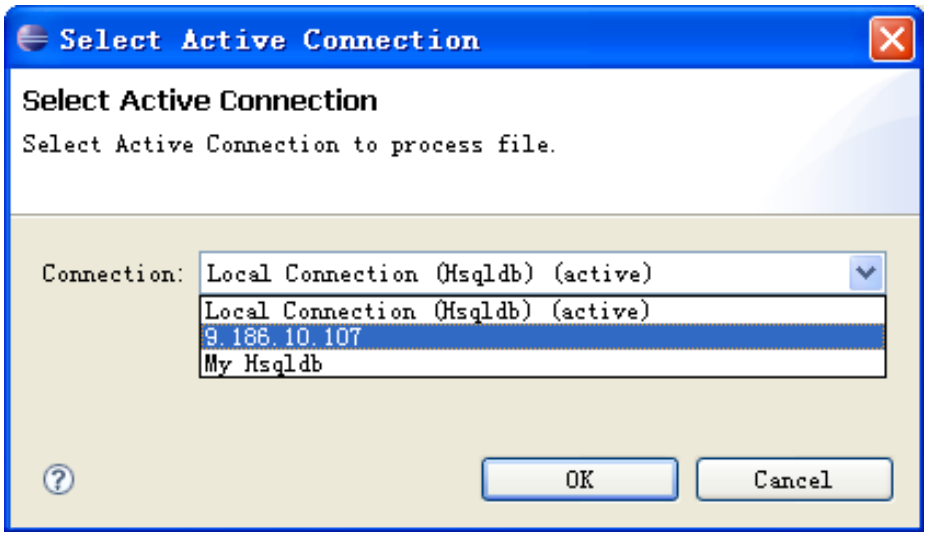


Set Hslqdb's size limitation for Profile Analyzer Support

You can also delete files manually to release disk space. Multi-selecting is allowed. See the picture below.



To set one Profile Analyzer support as active, you should switch the perspective to Profile Analyzer Perspective, so that the related menu named “Set Active” will occur, click it to set the connection you like as active.



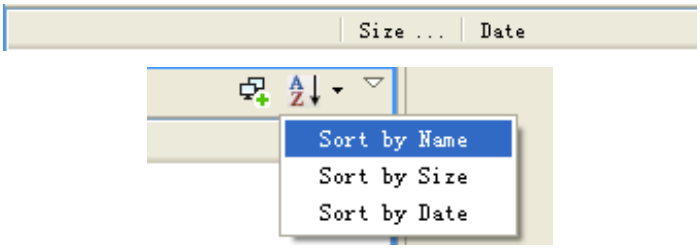
Select Active Connection

Active Profile Analyzer Support cannot be deleted. To delete it, you should set other support as active first.

Connection cannot be deleted, until you delete all the product supports under this connection.

This view support sorting operation. You can sort the files by name, size or date.

To sort, select the sort mode on the action bar, or click the title directly.



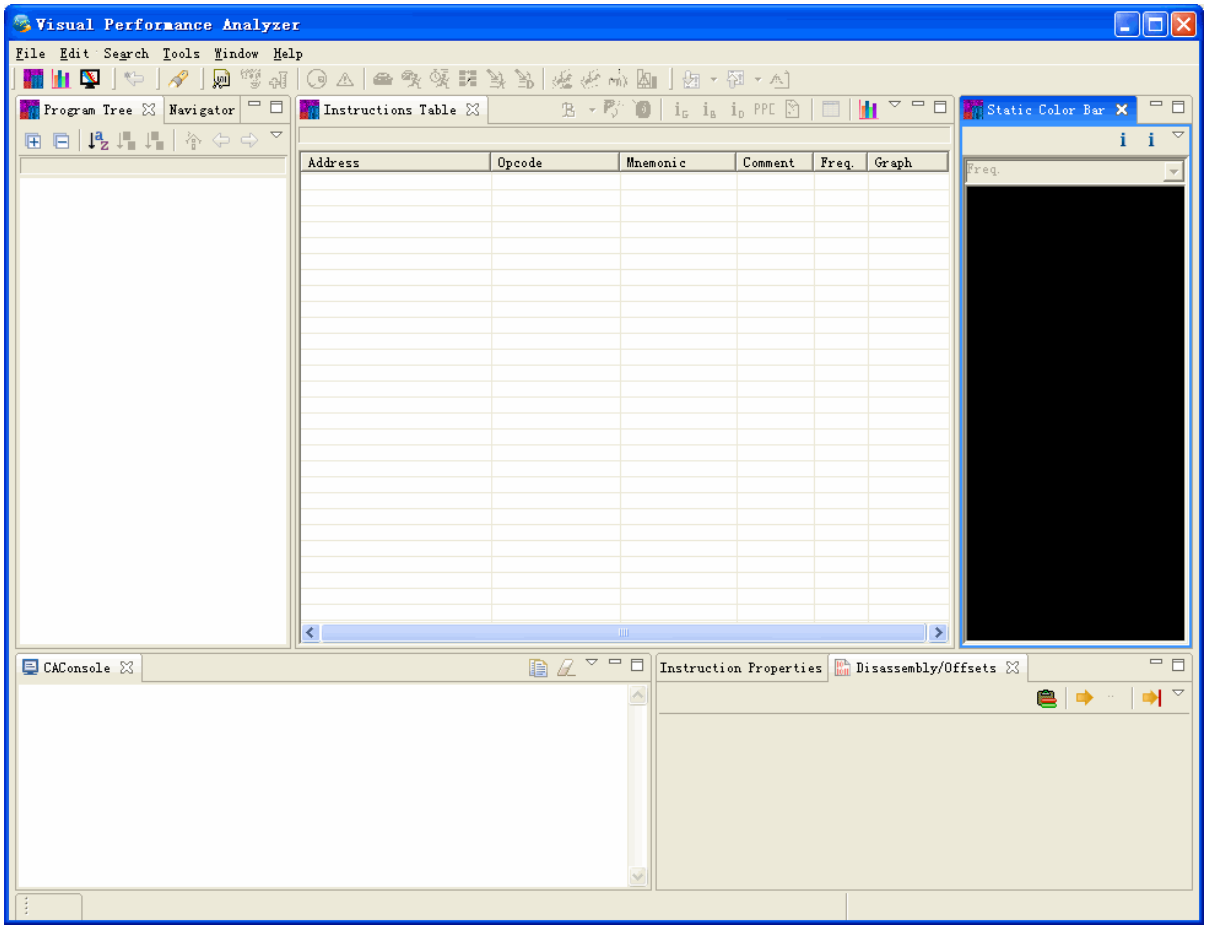
5.2Code Analyzer

You can also find the Code Analyzer User Guide from within VPA. Select **Help - Help Contents** within VPA. To get context sensitive help, press **F1** for Windows and AIX or press **Ctrl+F1** for Linux.

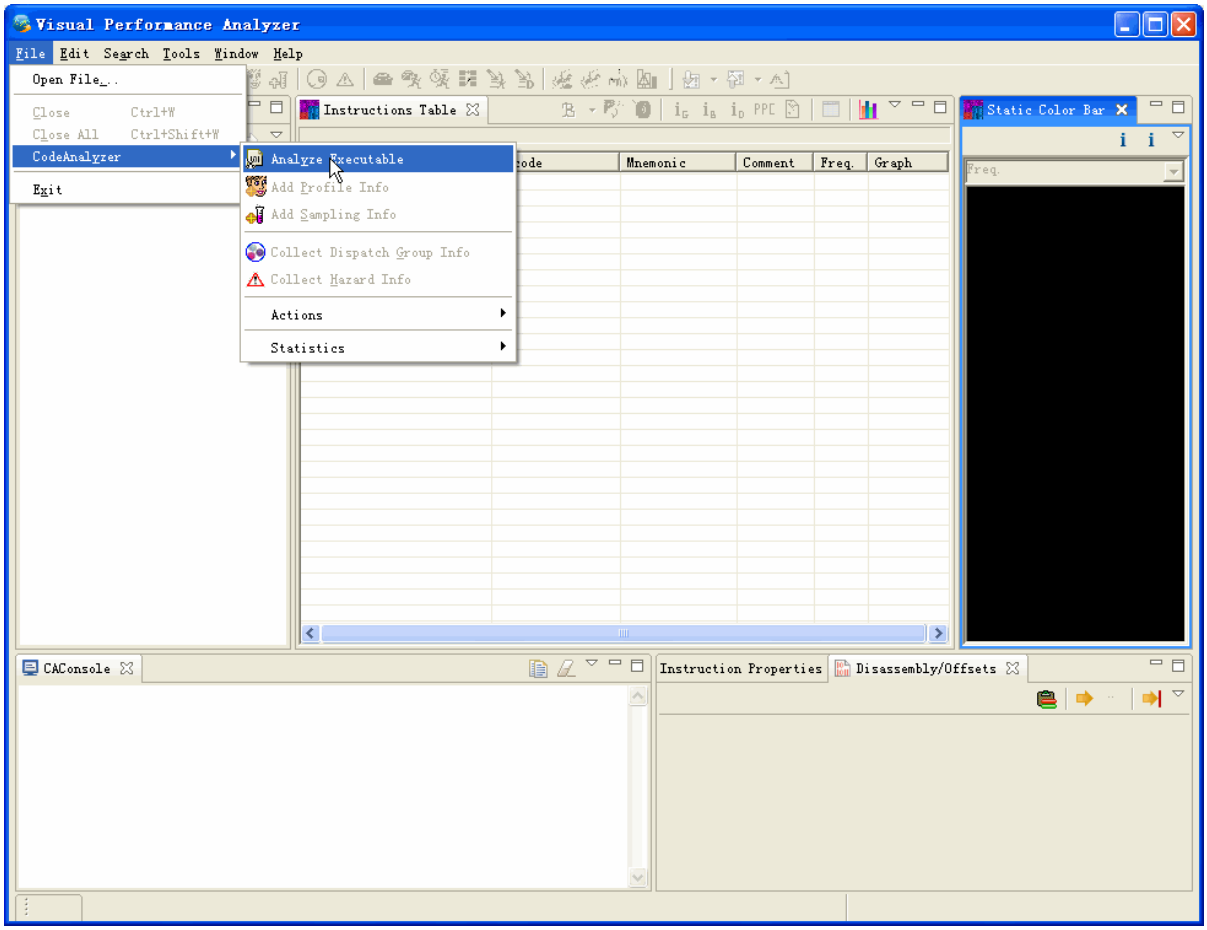
5.2.1Load an executable for analysis

When you first start Visual Performance Analyzer after installation, the default perspective is Profile Analyzer. To open CodeAnalyzer, you can choose **Windows -> Show Perspective -> Other-> CodeAnalyzer**.

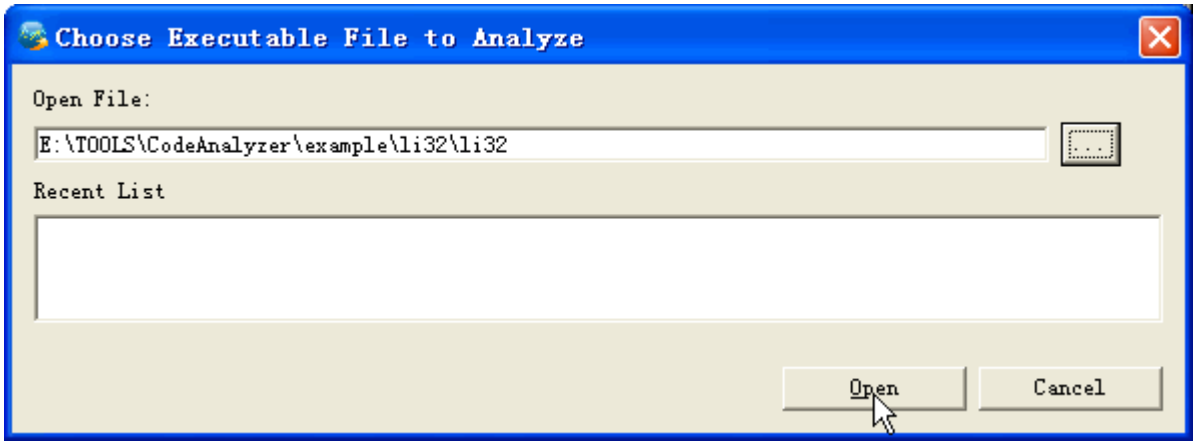
The following screen capture shows the default workbench window of CodeAnalyzer.



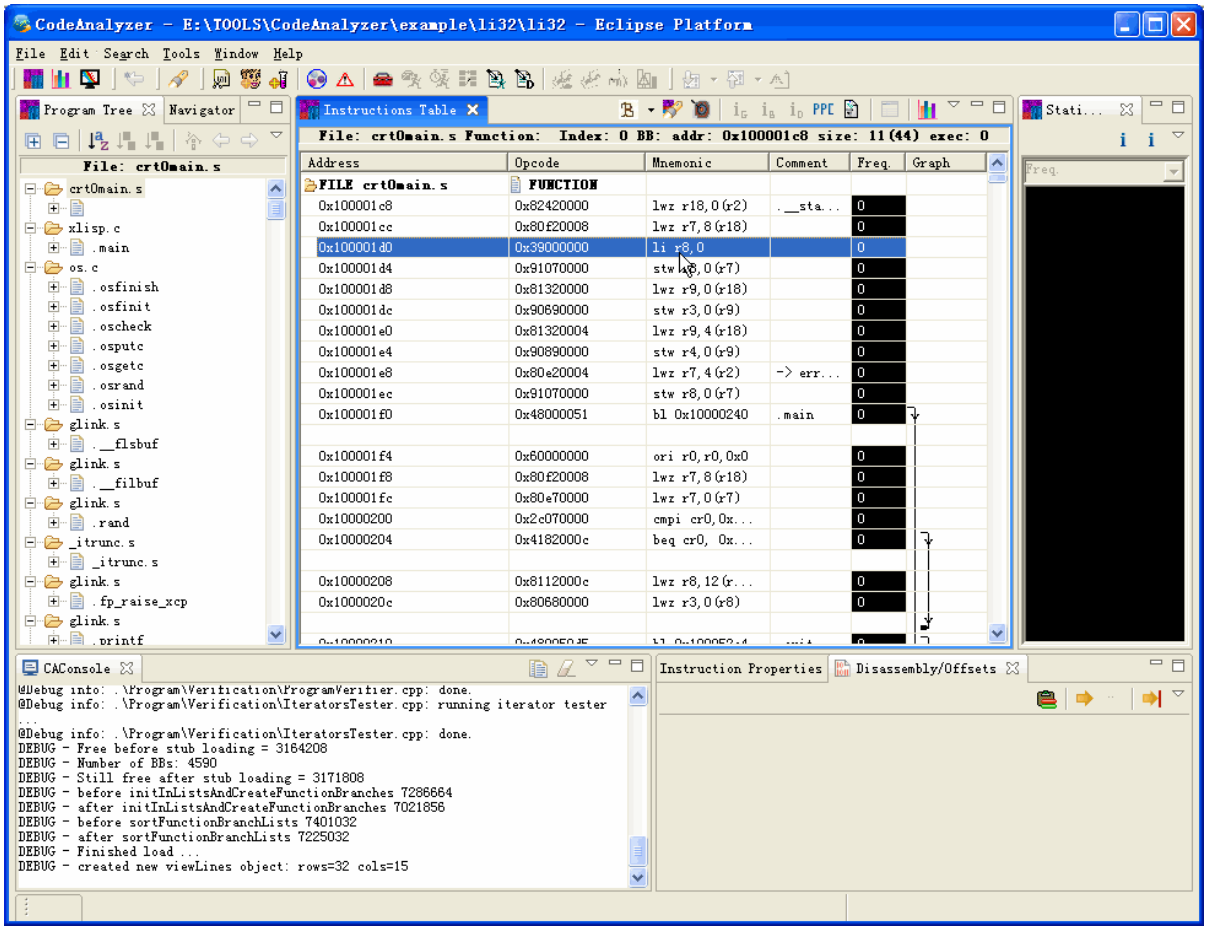
Choose **File -> CodeAnalyzer -> Analyze Executable**.



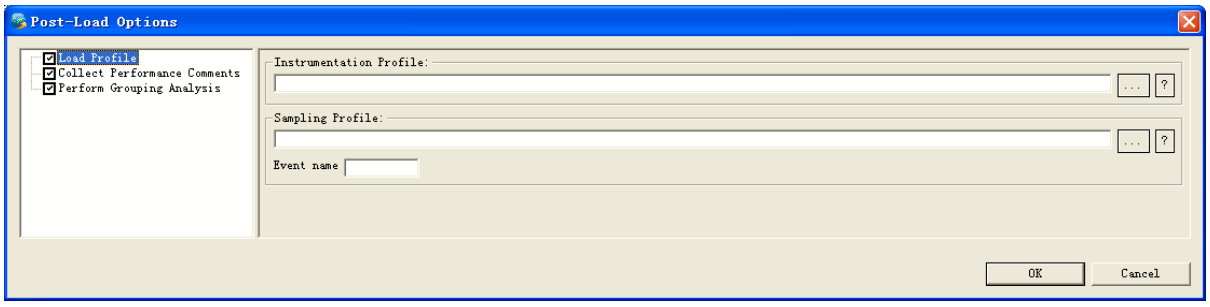
In the pop-up wizard, select the executable you want to analyze.



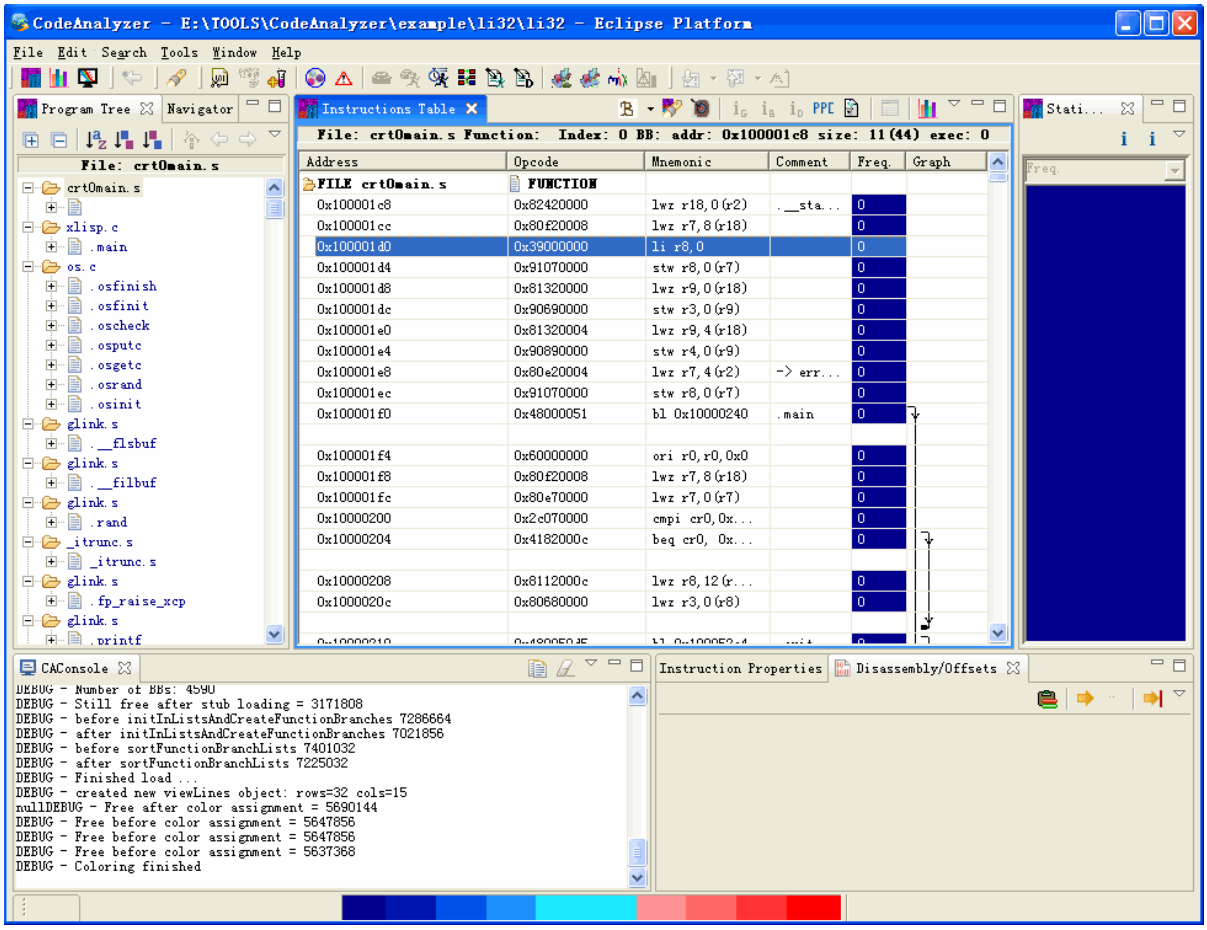
If you press **Open**, the executable will be loaded in CodeAnalyzer.



When an executable is loaded, a wizard for further action appears as follows that allows you to load profile information.



If you want to open profile information of the loaded executable, you can choose the address of profile file in the above wizard. After pressing **OK**, the profile information will be added into CodeAnalyzer workbench window.

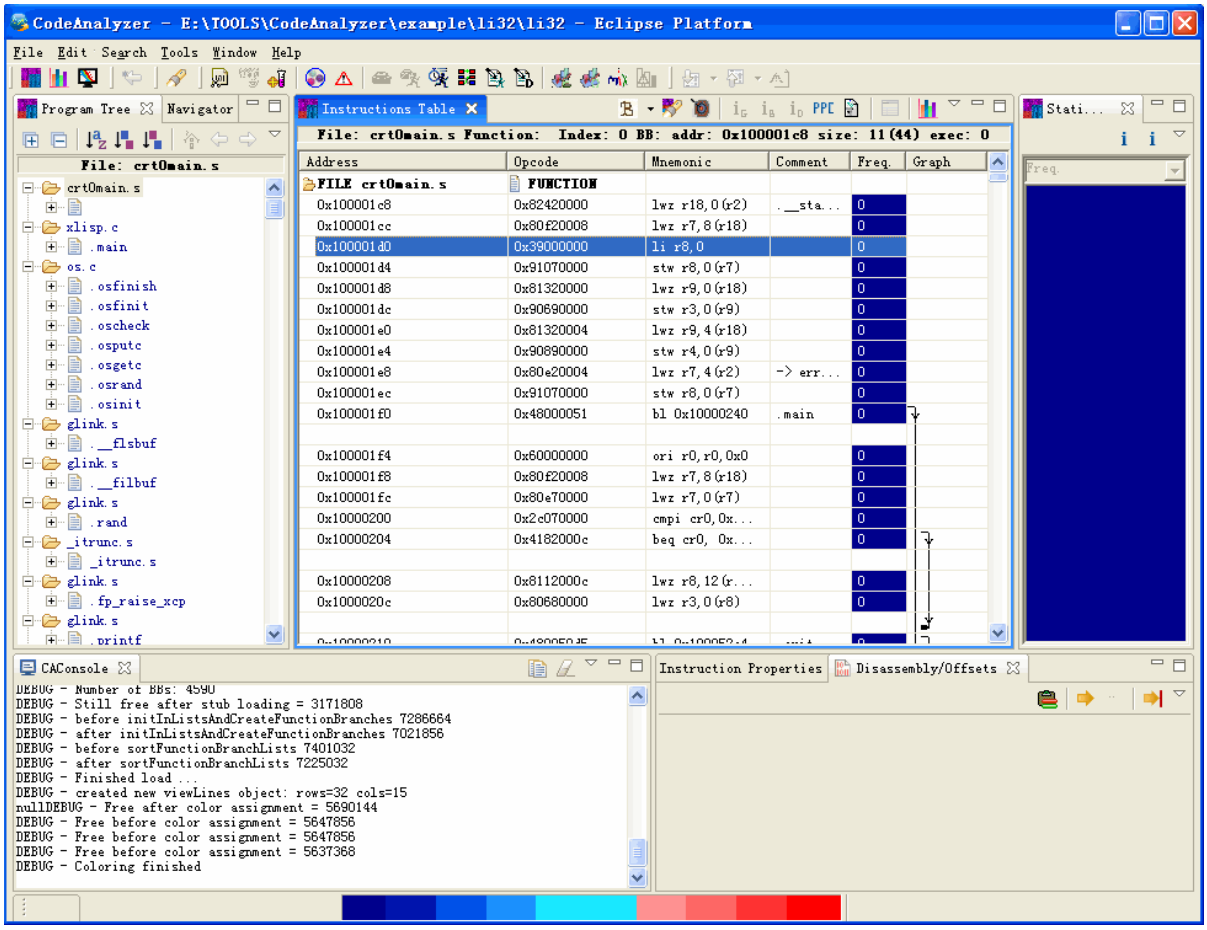


5.2.2Adding profiling information

If you have been working with an executable, without profiling information, you can add the profiling information by Choosing **File -> CodeAnalyzer -> Add Profile Information**.

If you find the profile information file, of the loaded executable, you can choose profile file in the wizard. After pressing **OK**, the profile information will be added into CodeAnalyzer workbench window.

The following screen capture shows the workbench window after profile information is loaded.



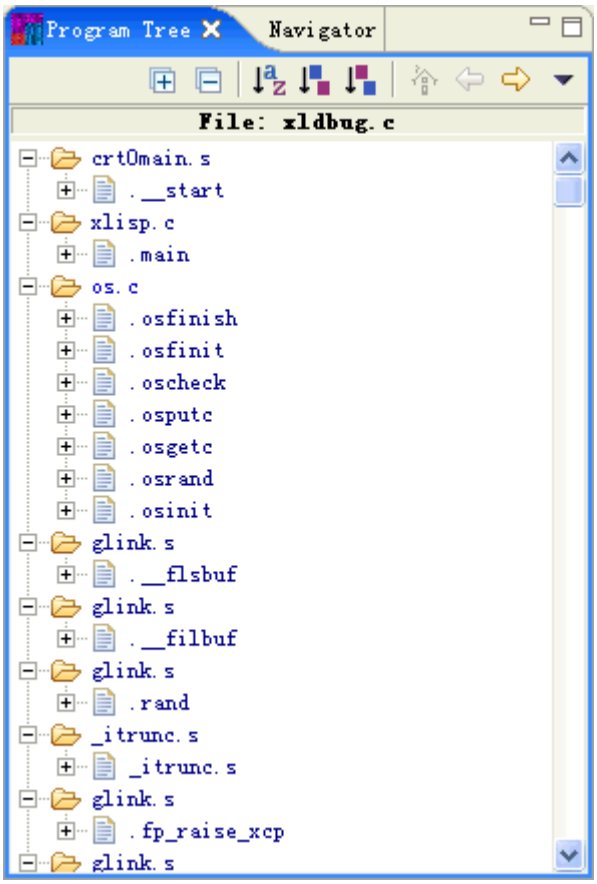
5.2.3Navigate the Executable

5.2.3.1Navigate the Program Tree

Program tree displays a hierarchical view of the loaded executable. It is automatically opened in the left side of the workbench window when you first use CodeAnalyzer. You can open program tree view by choosing **Windows -> Show View -> Program Tree**. First, you need to load an executable. Then navigate program tree by expanding all, collapsing all, sorting, going into opening source code and getting control flow.

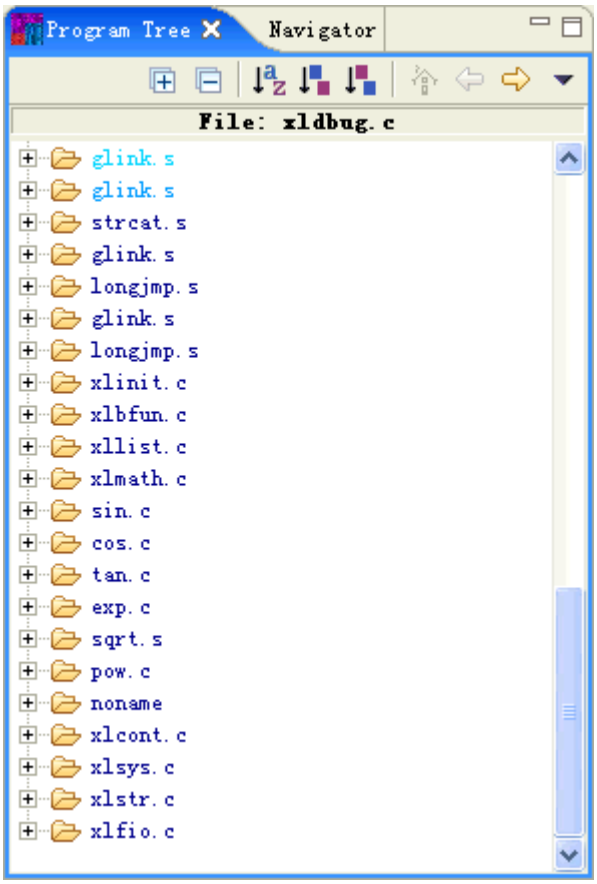
5.2.3.1.1Expand All

To get all the functions under each file in the loaded executable, press button in title bar. The result view of program tree expands as follows:



5.2.3.1.2Collapse All

To close all sub-items of each file in the executable, press button in title bar. The result view of program tree shows as follows:



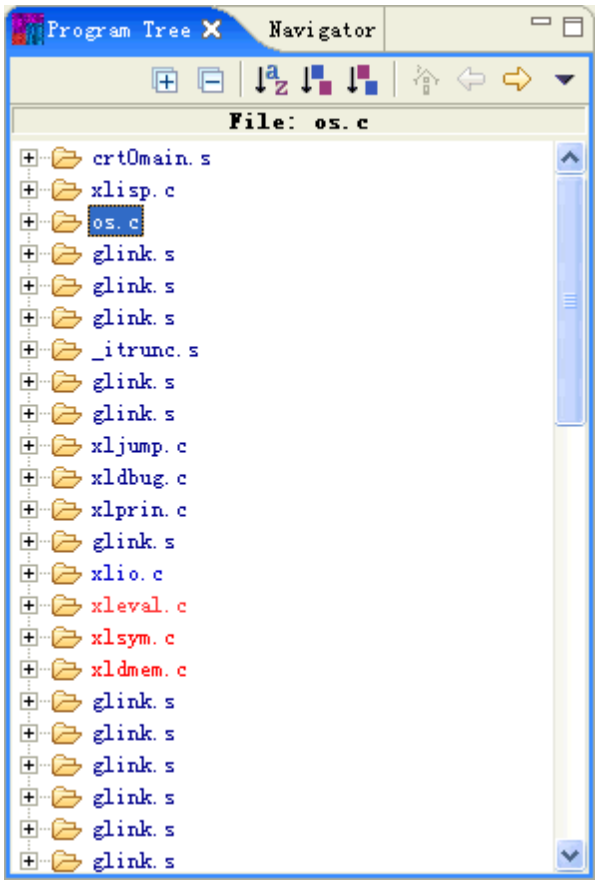
5.2.3.1.3Sort

There are three kinds of sorting in program tree view: lexicographical order, ascending order and descending order. You can press their corresponding buttons , , , in the view's title bar or right-click any object in the view and choose these items.

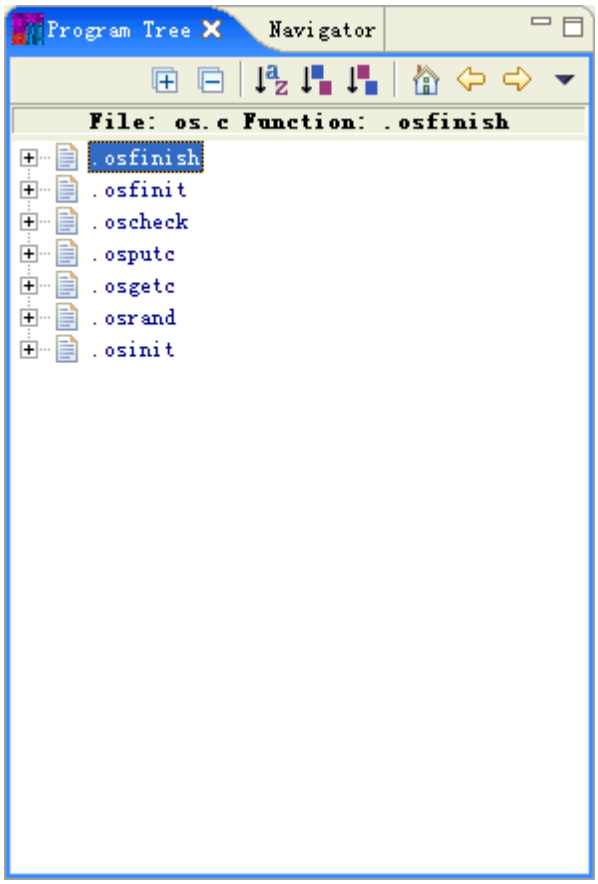
5.2.3.1.4Go Into

If you want to get detailed information of the items which are under certain object, you can select this object and press . The program tree will display all its sub-items. If you want to go to upper level, just press . Pressing can lead you to root view.

For example, if you want to see all the functions in file os.c, choose os.c file in program tree as follows:



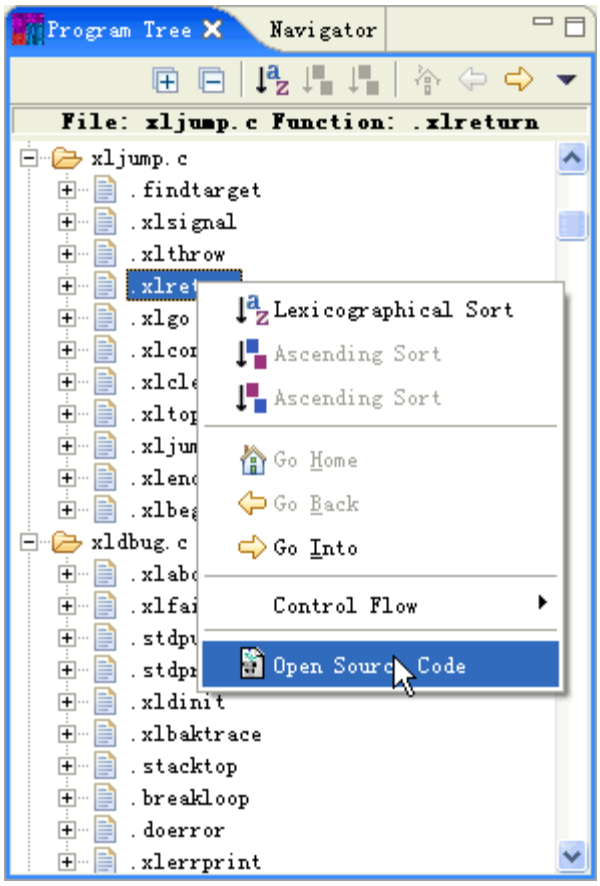
The result view displays all the functions within this file.



You can also navigate forward or backward as you like.

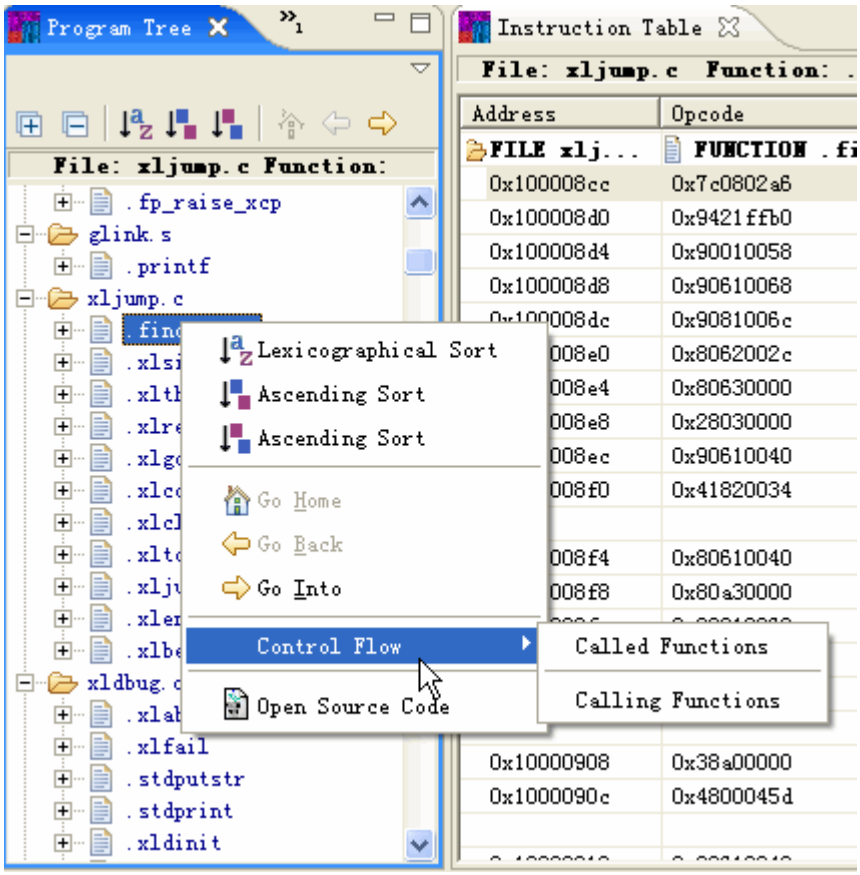
5.2.3.1.5Open source code

To open source code, right-click any object in program tree and select **Open Source Code**.

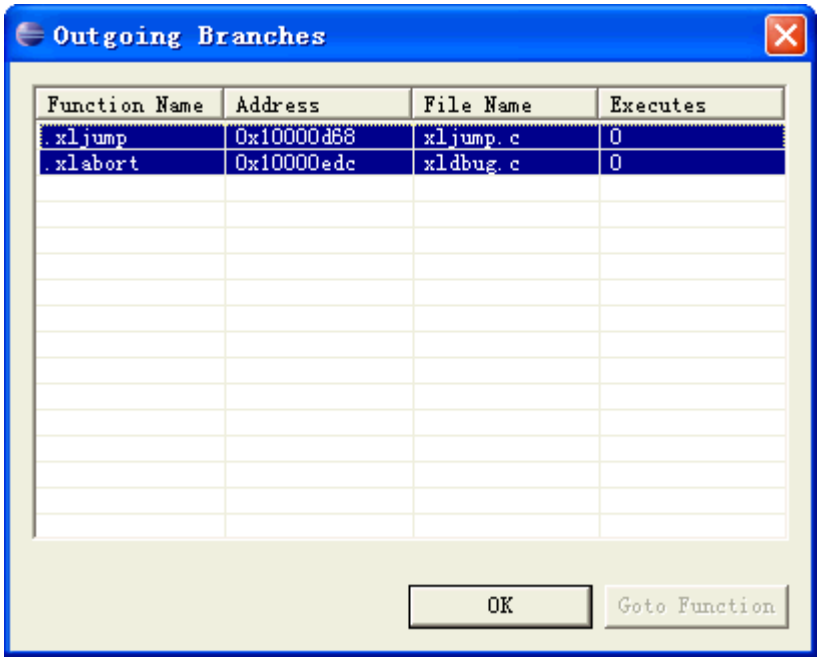


5.2.3.1.6Get Control Flow

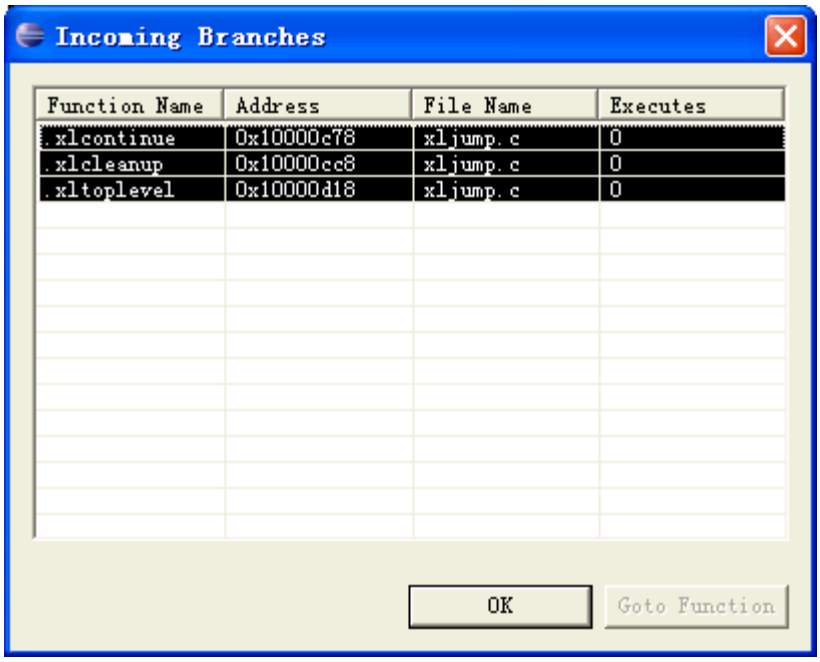
To get calling functions and called functions, select a function in the program tree, right-click and choose **Control Flow**.



The called functions of the selected function in the above screen capture show as follows:



The calling functions of the selected function in the above screen capture show as follows:



5.2.3.2View Static Color Bar

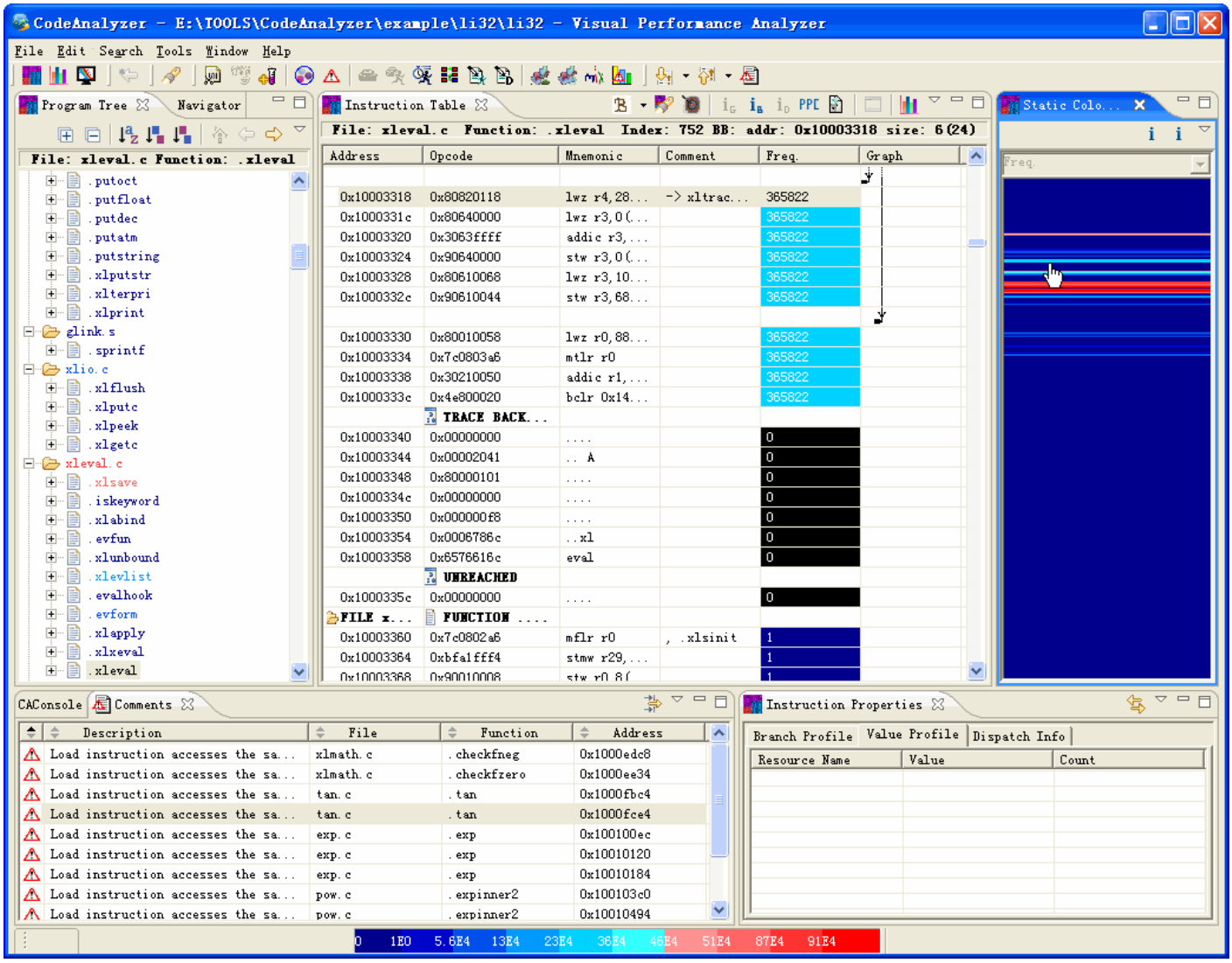
Static color bar gives an overview of frequency distribution of basic blocks in the loaded executable. You can open this view by choosing **Windows -> Show View -> Static Color Bar**.

To obtain this information, do following steps:

1. Load an executable
2. Add profile information

After profile information is added, there is a yellow pointer in the static color bar. It indicates the position of basic block you have selected. The yellow pointer scrolls in accordance with the basic block you select in the **Instructions Table** or **Program Tree**.

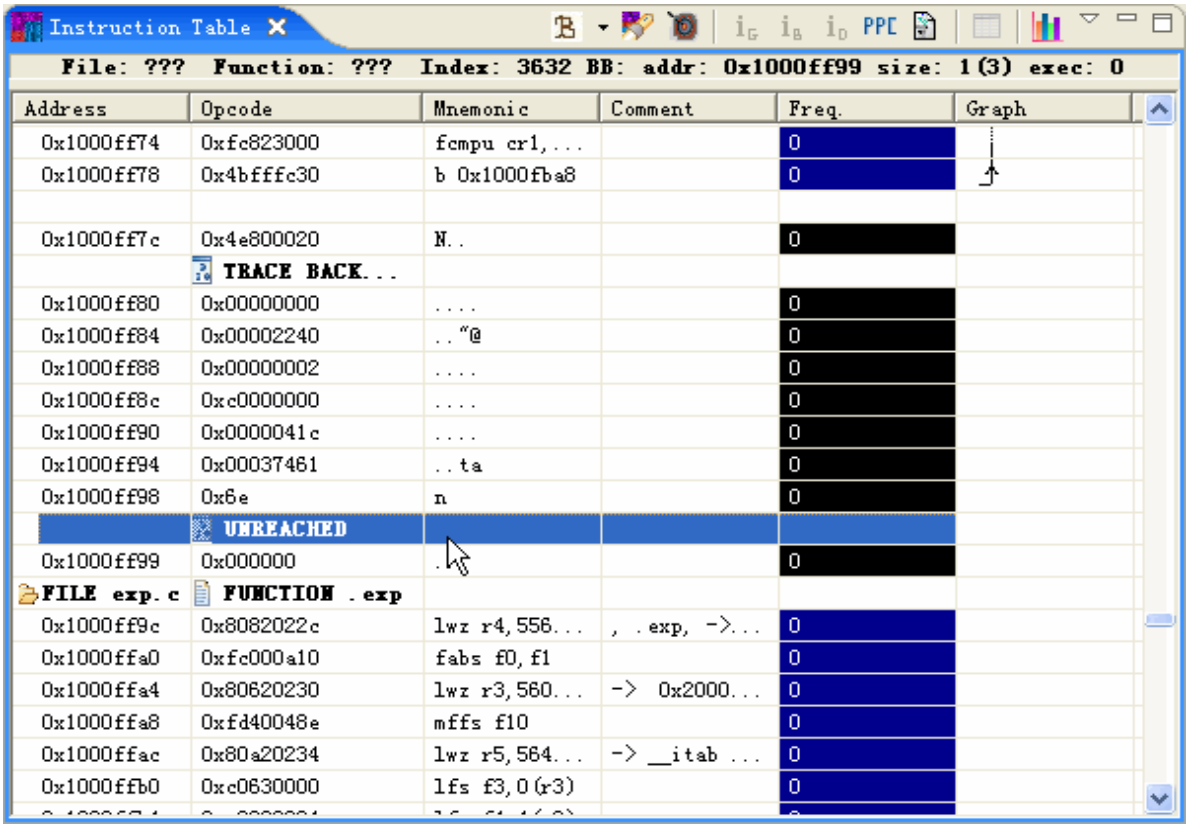
The following screen capture shows the selection of a specific color bar.



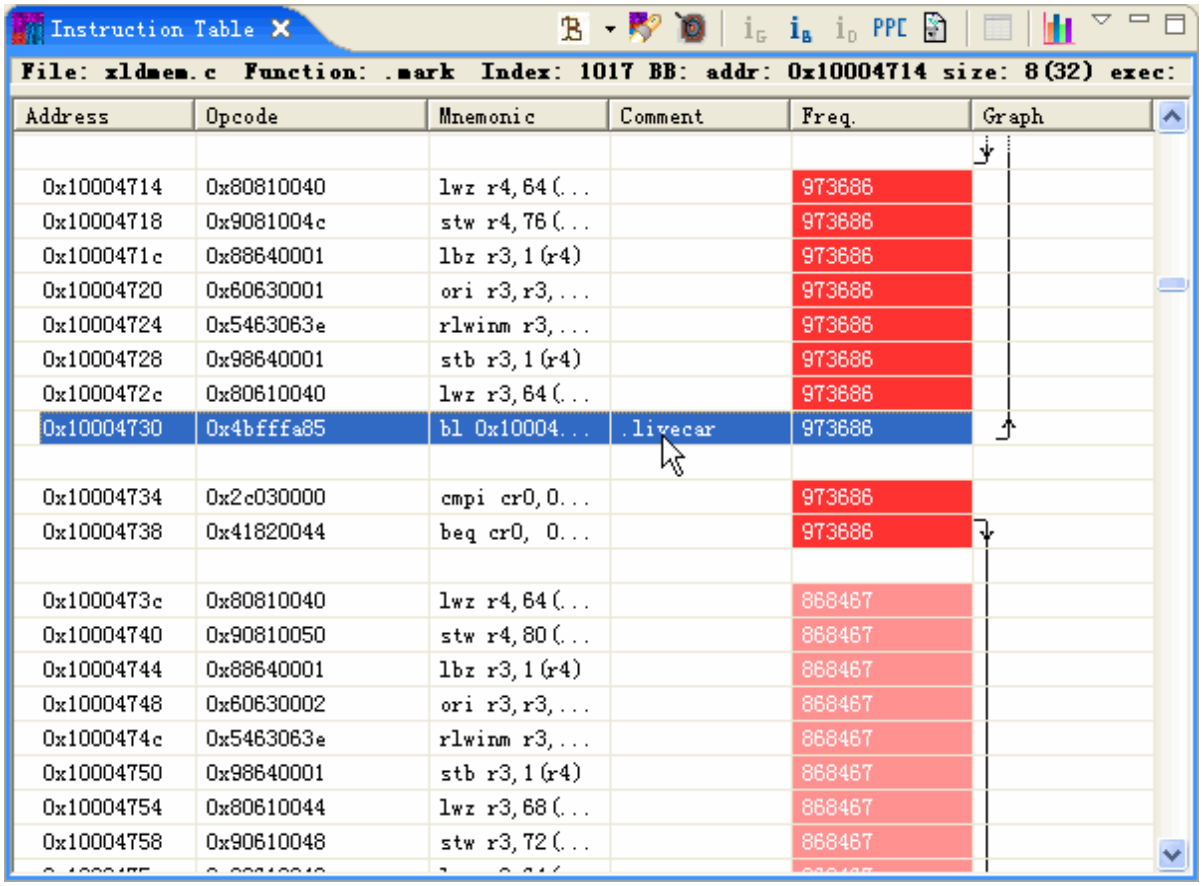
5.2.3.3Navigate Instructions View

Instructions view is the default view of Instructions Table. It shows the contents of an executable or shared object as a table of assembly instructions, with its control flow graph drawn vertically at its side. To open this view, select **Switch to instructions** in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table.

In instructions view, when a new file begins, there is a line of table beginning with icon and followed by its name. When a new function starts, there is also declaration in a separate line. Instructions which belong to the same basic block are organized together with blank lines between them. The last basic block in a function begins with a line as follows.



After adding FDPR-Pro profile information, you can get the frequency of each instruction in color. The denotation of each kind of color can be found in the lower part of workbench window. Instruction groups are indicated in the front of each instruction. In **comment** column, there are red triangles which contain performance comments to specific instructions.



If you right-click any line item in Instruction View, a menu list appears. Typically, there are four sections of this menu. In the first section, you can get PPC(Power PC) assembly reference help, branch profile information, dispatch information, and set points to collect the value of important resources during profiling. In the second section, you can choose to find specific instruction or open source code of the selected instruction. In the third section, the menu will show the target of the basic block (**Fall thru** means the next basic block) to which the selected instruction belongs. The last section shows the callers of the basic block to which the selected instruction belongs. The first and the last two sections may be varied according to the selected instruction.

5.2.3.3.1Get PPC help

To get the assembly reference of certain instruction, right-click on the instruction and choose Show PPC Help. You can also select the instruction and press PPC button in title bar.

5.2.3.3.2Show Dispatch Information

To get the dispatch information of certain instruction, right-click this instruction and choose **Show Dispatch Info**. You can also select this instruction and press i_G button in title bar. To obtain more detailed description, please refer to View dispatch information.

5.2.3.3.3Show Branch Information

To get the branch information of certain basic block, right-click the last instruction of this block and choose **Show Branch Info**. You can also select the last instruction of this basic block and press i_B in title bar. To obtain more detailed description, please refer to View branch profile.

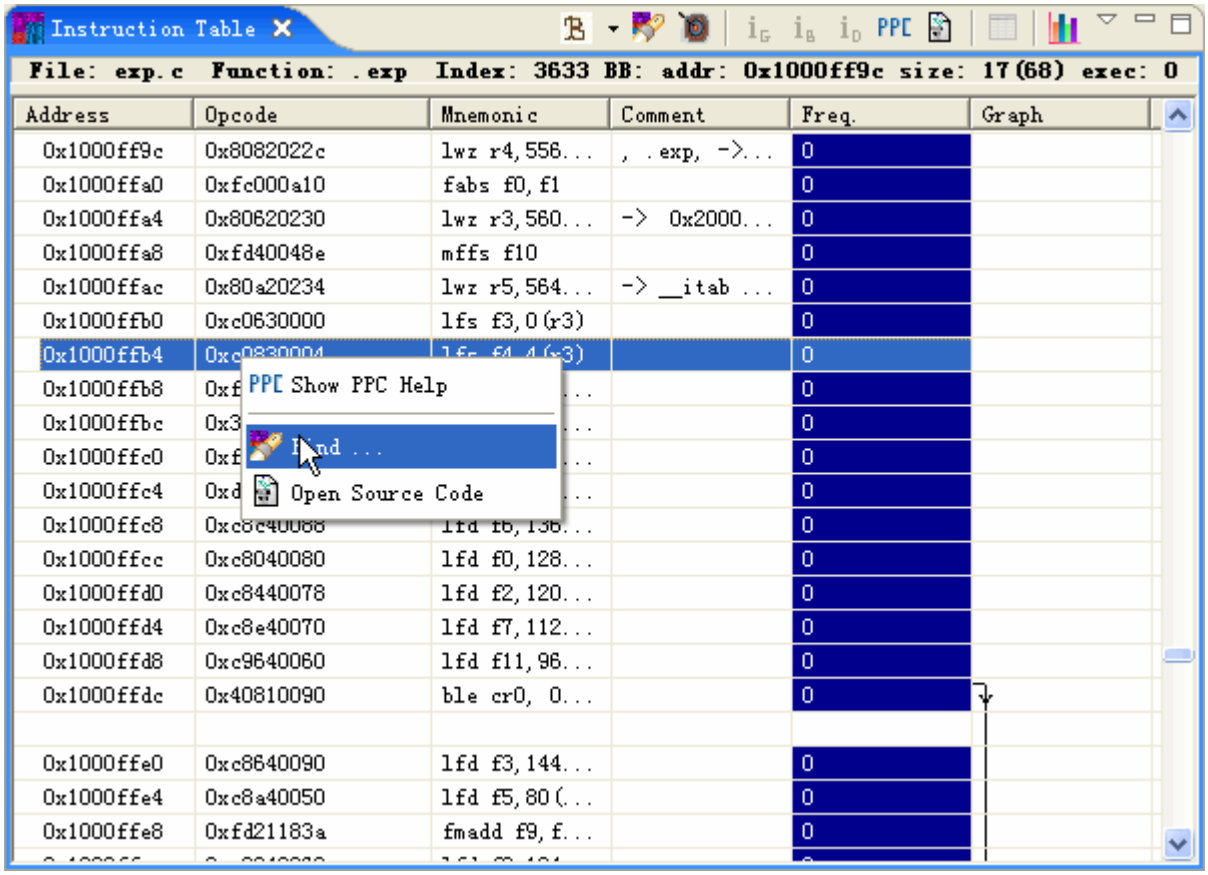
5.2.3.3.4Collect Value Profiling

Before profile information of the loaded executable is added, you can collect value of certain resources of specific instructions. To get this information, you need to first select these instructions by right-clicking an instruction and choosing **Collect Valuing Profiling**. A wizard will appear for you to choose resources value you try to get. You can also do this by pressing button in title bar. To obtain more detailed information, please refer to View value profile.

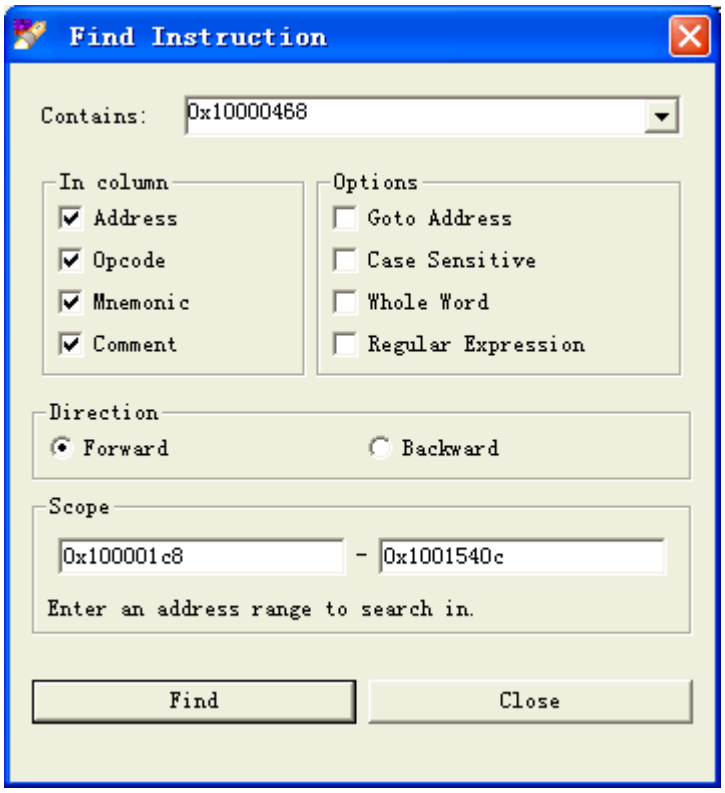
5.2.3.3.5Find Instructions

To find specific instructions in this view, press button in title bar. You can also open it by right-clicking any instruction and choosing **Find**.

The following screen capture shows the initial activating of Find wizard.



A Find Instruction wizard appears.



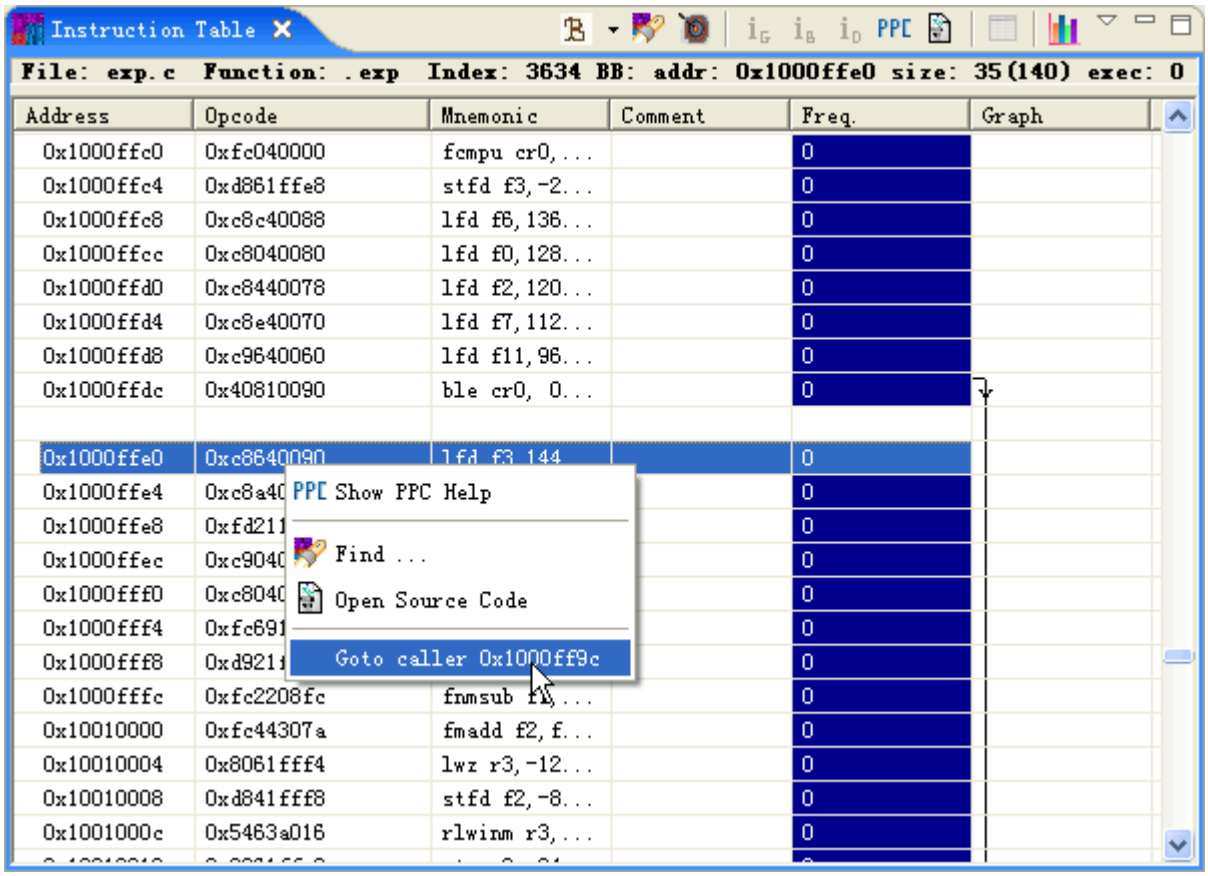
The default instruction to be sought is that of the selected instruction. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select **Forward** or **Backward**. Error and information messages are displayed in the bottom of the dialogue.

5.2.3.3.6Open Source Code

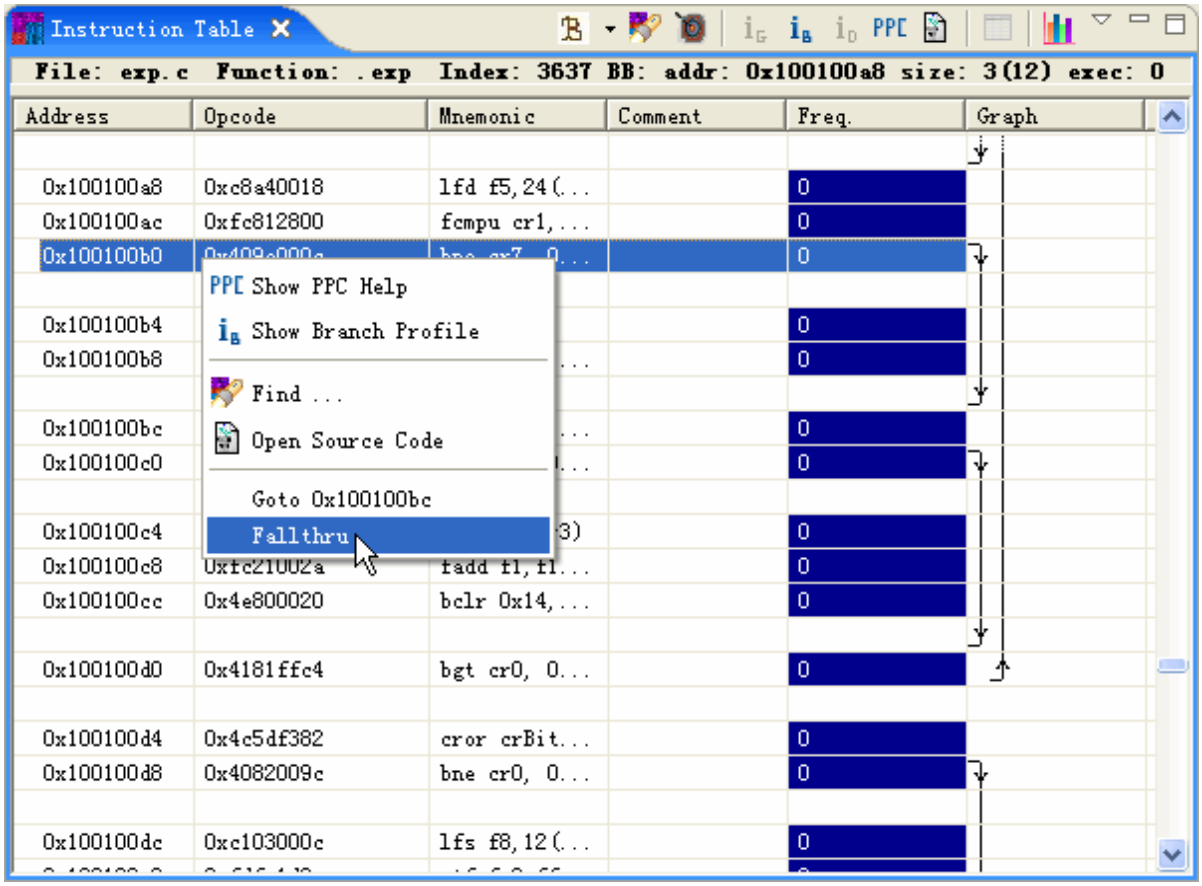
To get the source code of the selected instruction, right-click this instruction and choose **Open Source Code**. You can also use the button in title bar. For more information, please refer to View source code.

5.2.3.3.7Go to callers and callees

Right click the first instruction in this basic block. The menu list shows its callers, their addresses and functions name which they belong to. The caller is defined to be the first instruction of basic block which calls the selected instruction.



To get target basic blocks of the selected instruction, right-click the last instruction of a basic block. The menu list will show its target basic blocks, with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows **Fall thru** only.



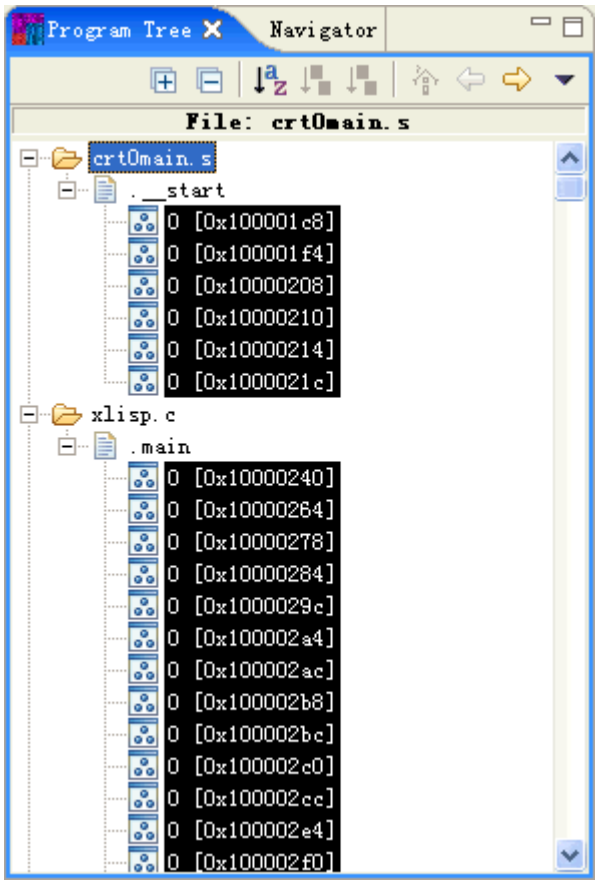
The screen capture above shows that the selected basic block has two target basic blocks. One is the one below, and the other is in the function .main. You can jump to these basic blocks simply by clicking it. You can verify this relationship by referring to graph beside it.

5.2.3.3.8Navigate along with program tree

In the program tree on the left side of the workbench window, a hierarchical organization of loaded executable or shared object is displayed. You can navigate along Instructions Table by selecting objects in program tree, or vice verse.

For example, if you select the first file in the program tree, the first instruction of this file will be highlighted in instructions view, the first basic block of this file will be highlighted in blocks view and the first function of this file will be highlighted in functions view.

The following screen capture shows the selection of the first file in Program Tree.

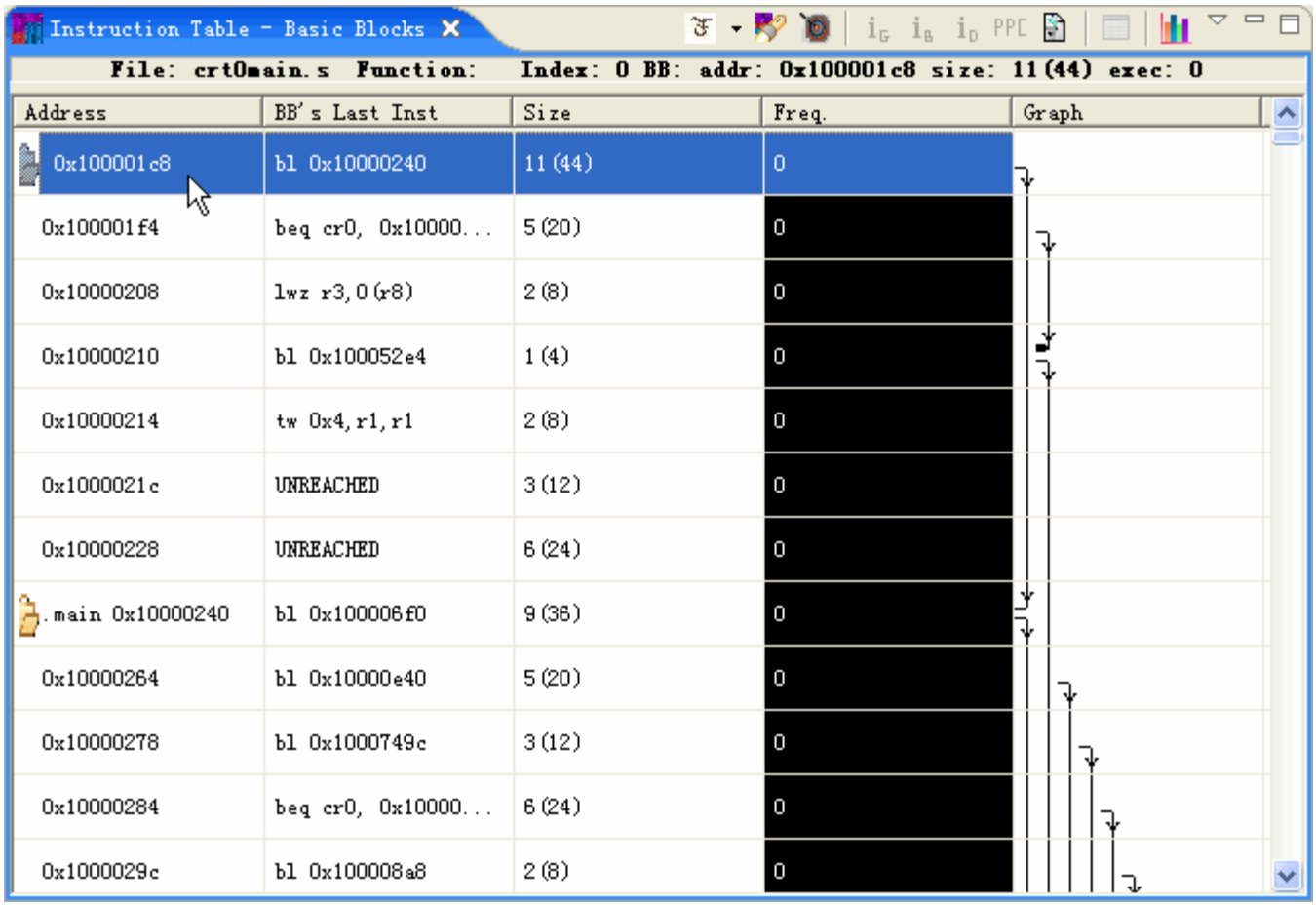


The selected instructions in instructions view changes correspondingly.

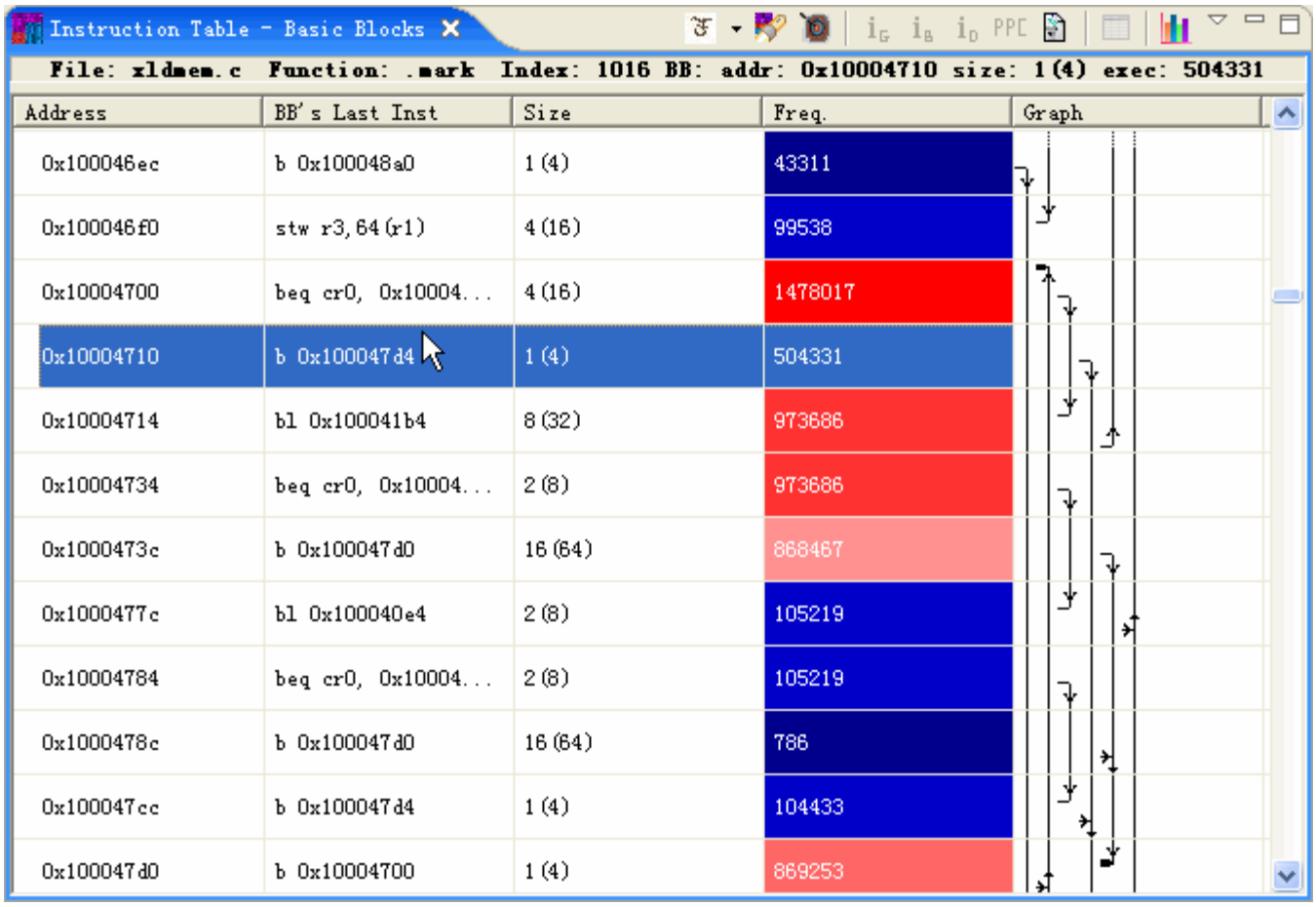
5.2.3.4Navigate the Blocks View

Blocks View shows the detailed information of loaded executable or shared object in form of basic blocks. To open this view, select **Switch to blocks** in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table .In blocks view, there is an icon in the front of the first basic block of a file in loaded executable or shared object. When a new function begins, an icon appears in the front of its first basic block.

The following screen capture shows blocks view of a loaded executable.



After adding FDPR-Pro profile information, you can get the frequency of each basic block in color. The denotation of each kind of color can be found in the lower part of the window.

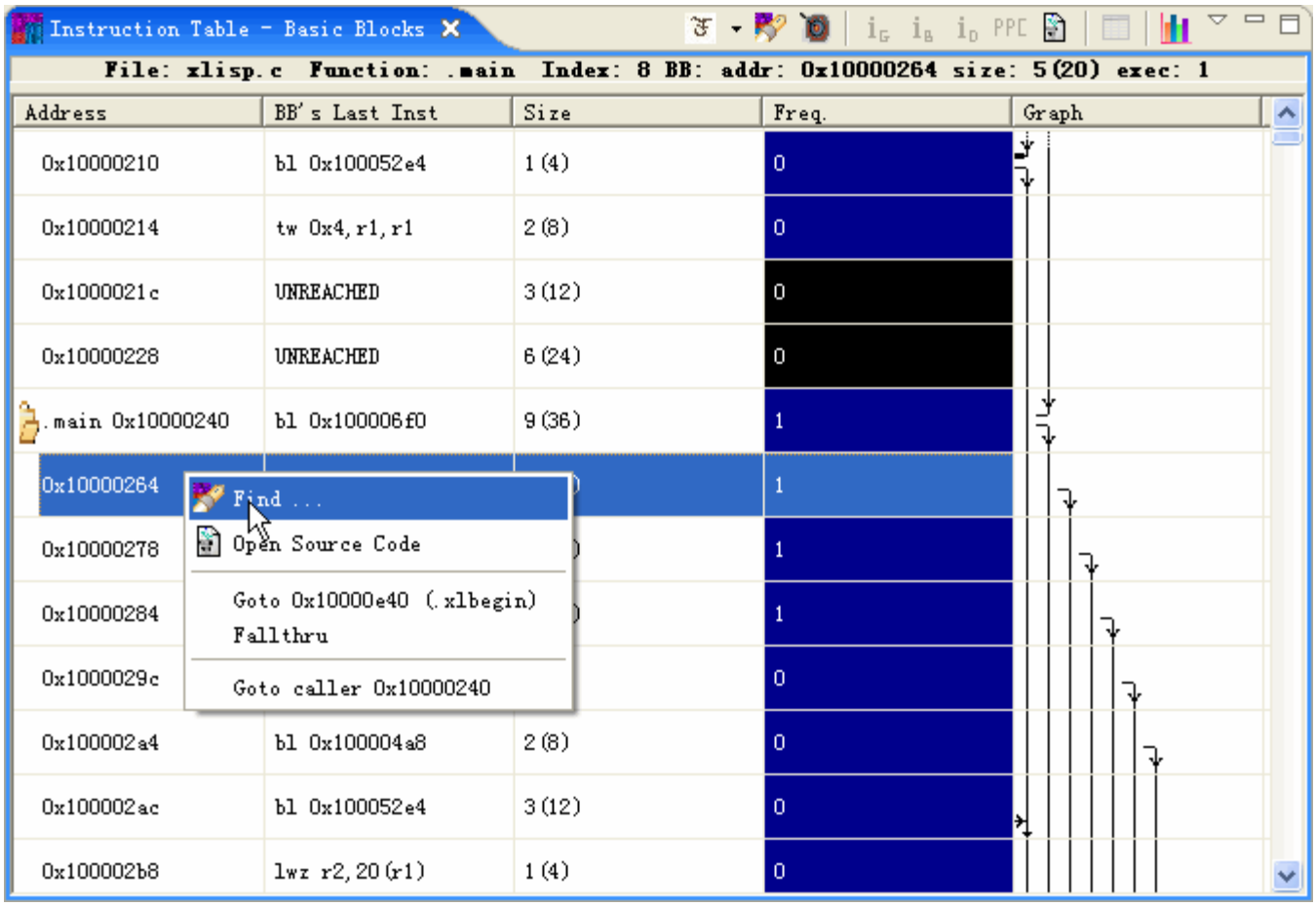


If you right-click a basic block, a menu list will appear. Typically, there are three sections in this menu. In the first section, you can search for specific basic block or open source code of the selected basic block. In the second section, the menu shows the target of this basic block (**Fall thru** means the next basic block). The last section gives information of all the callers of this basic block. The last two sections may be varied according to the selected basic block.

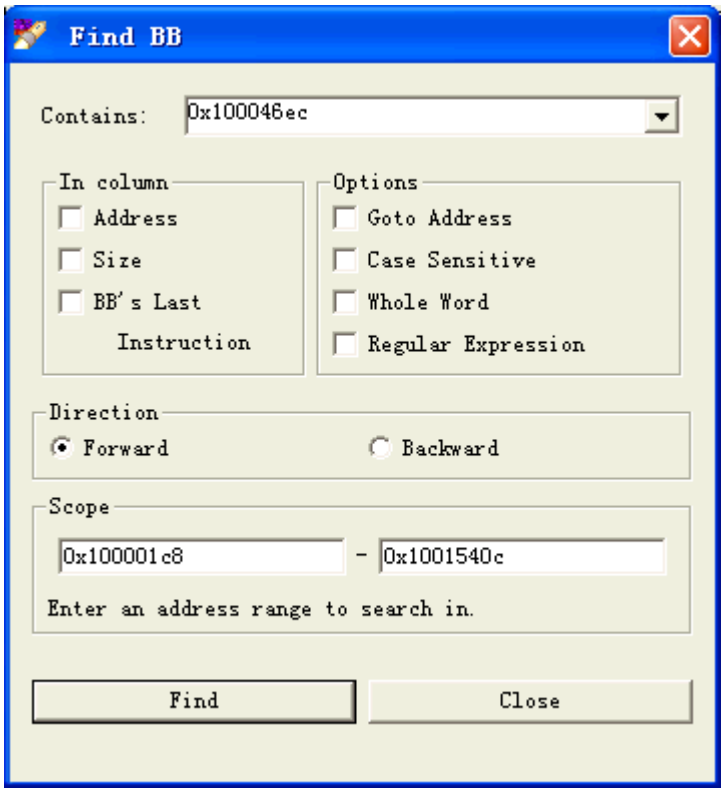
5.2.3.4.1Find basic blocks

To find specific basic blocks in this view, press button in title bar. You can also open it by right-clicking any basic block and choosing **Find**.

The following screen capture shows the initial activating of Find wizard.



A Find BB wizard appears.



The default basic block to be sought is that of the selected one. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select **Forward** or **Backward**. Error and information messages are displayed in the bottom of the dialogue.

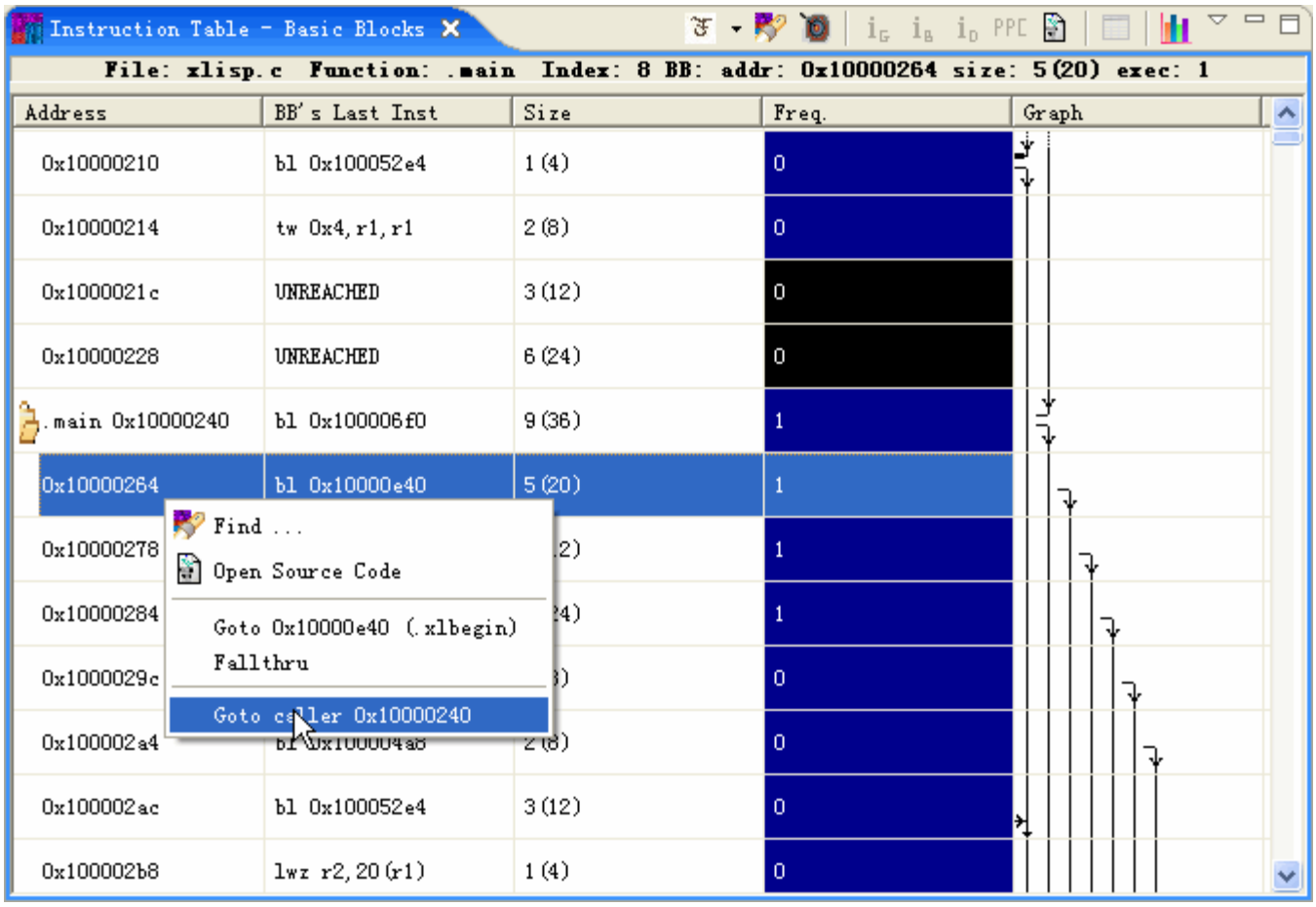
5.2.3.4.2Open Source Code

To get the source code of the selected basic block, right-click this basic block and choose **Open Source Code**. You can also use the button in title bar. For more information, please refer to View source code.

5.2.3.4.3Go to callers and callees

Right click any basic block in this view. The menu list shows all its callers and their addresses. The address of caller is defined to be that of the first instruction in the basic block which calls the selected basic block. The menu list also displays the information of its target basic blocks, along with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows **Fall thru** only. By choosing its callees or callers, you can jump directly to these basic blocks.

In the screen capture below, the selected basic block has two callees and one caller.

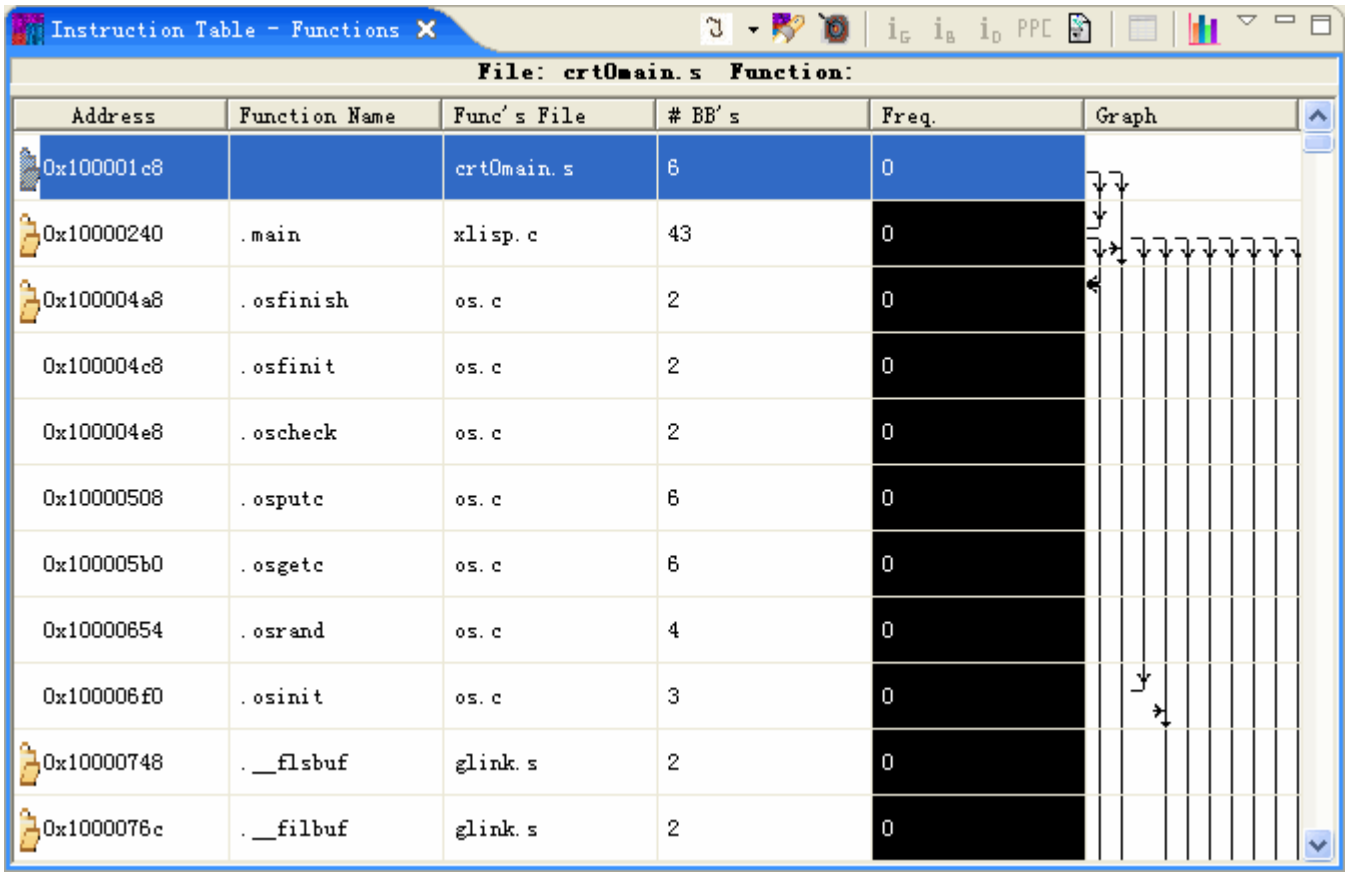


You can verify these relationships by referring to graph column on the right side.

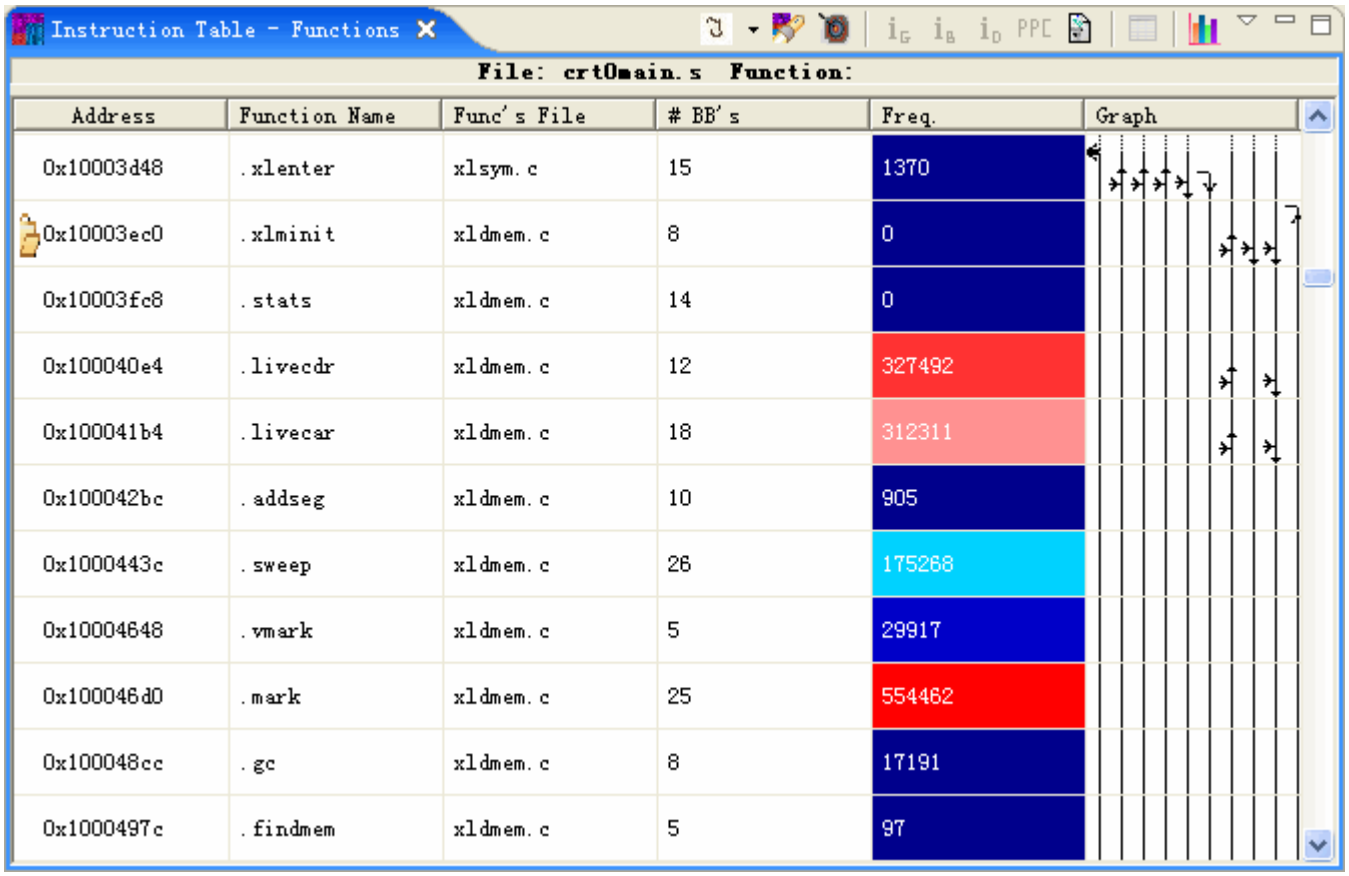
5.2.3.5Navigate Functions View

Functions View shows the detailed information of loaded executable or shared object in form of functions. To open this view, select **Switch to functions** in the Instructions Table's title bar. The explanation of functions of some buttons in title bar can be found in Instructions Table .In functions view, there is an icon in the front of the first function of a file in loaded executable or shared object.

The following screen capture shows functions view of a loaded executable.



After adding FDPR-Pro profile information, you can get the frequency of each function in color. The denotation of each kind of color can be found in the lower part of workbench window.

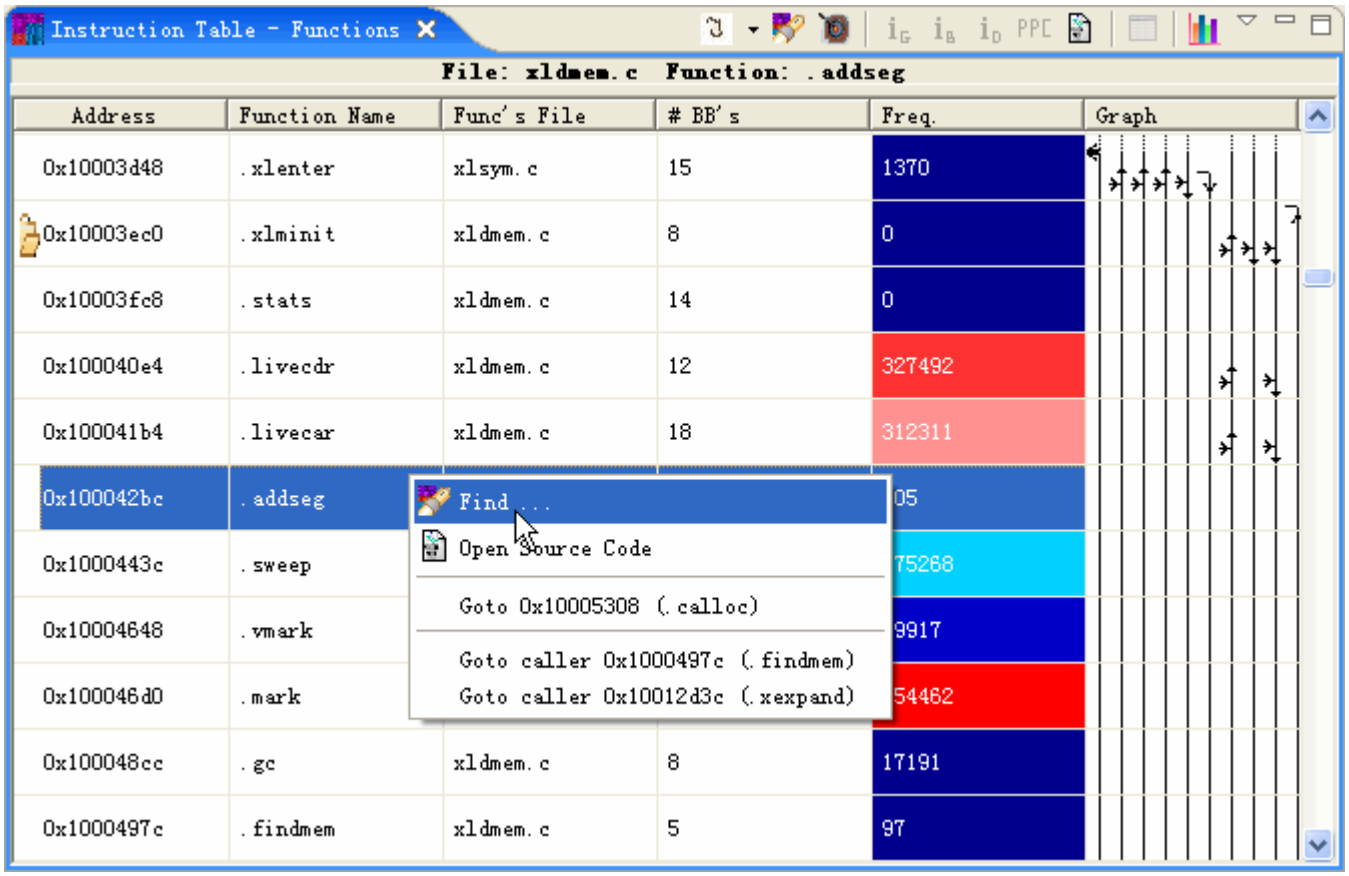


If you right-click a function, a menu list will appear. Typically, there are three sections of this menu. In the first section, you can choose to find functions or open source code of the selected function. In the second section, the menu shows the target functions of this one. The third section displays the callers of this function. The last two sections may be varied according to the selected function.

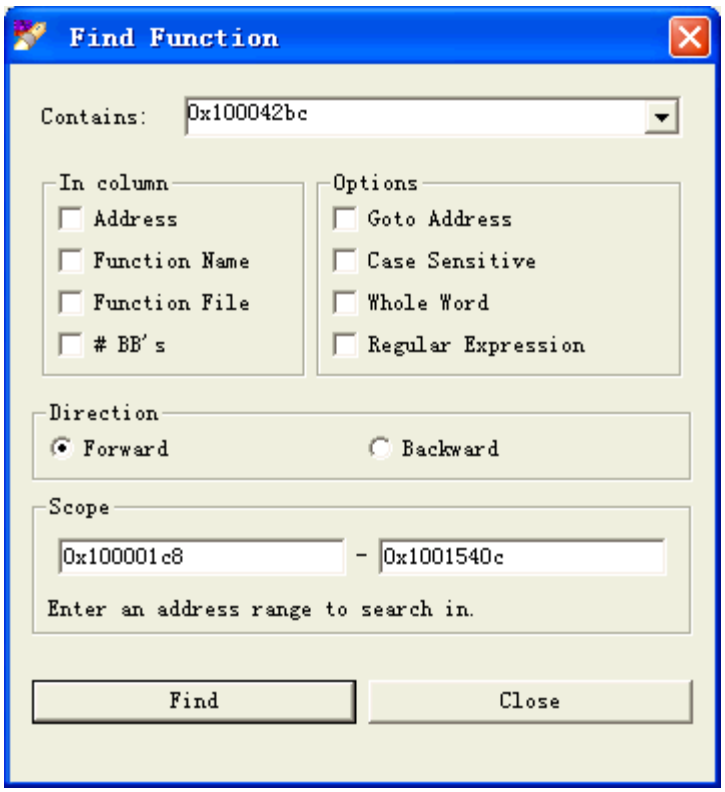
5.2.3.5.1Find functions

To find specific functions in this view, press button in title bar. You can also open it by right-clicking any function and choosing **Find**.

The following screen capture shows the initial activating of Find wizard.



A Find Function wizard appears.



The default function to be sought is that of the selected one. The initial scope of this wizard is set between the start address and the end address of the loaded executable or shared object. You may change the scope as you like, but it should not exceed the boundary. To choose the direction of searching, select **Forward** or **Backward**. Error and information messages are displayed in the bottom of the dialogue.

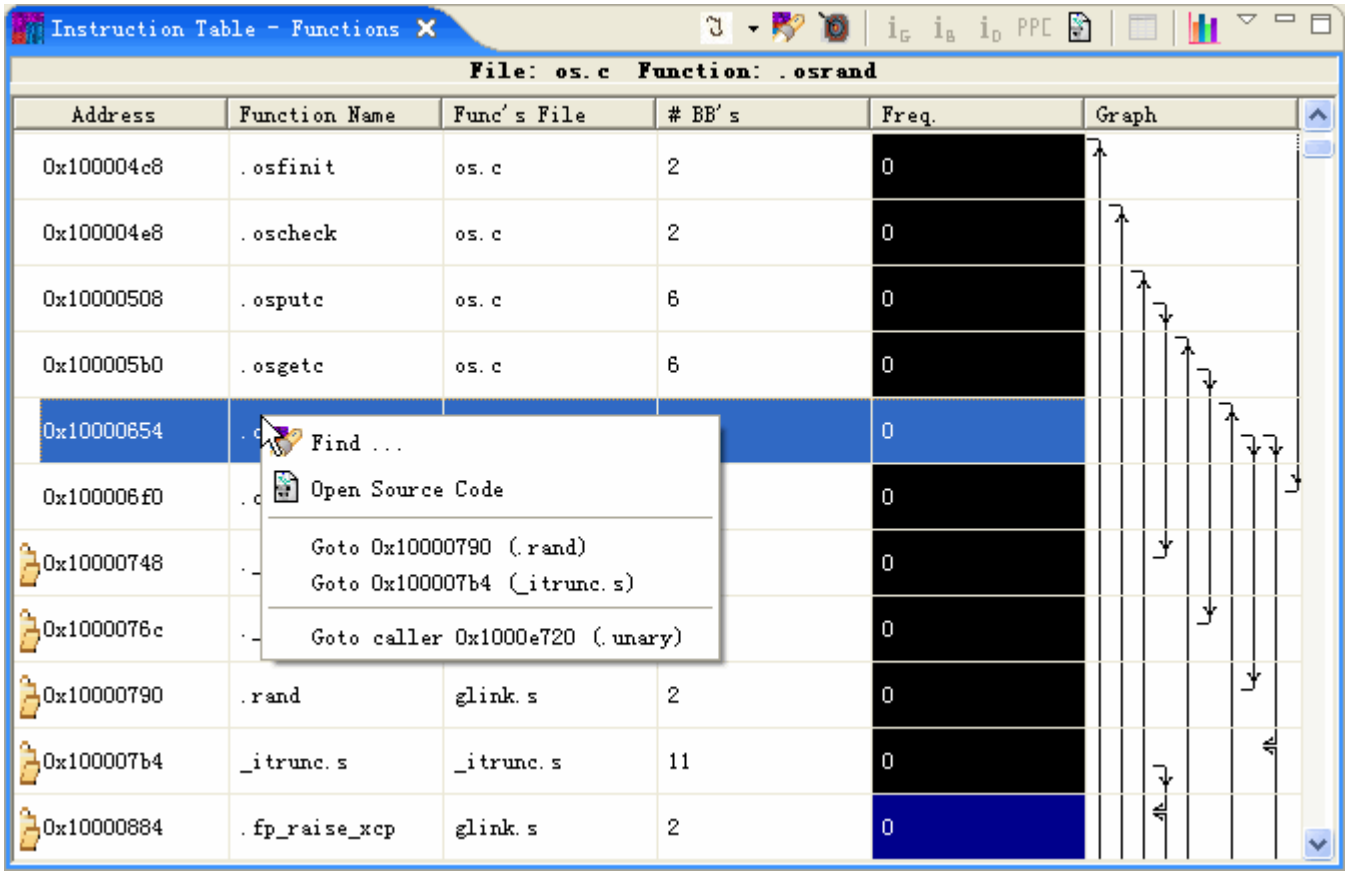
5.2.3.5.2Open Source Code

To get the source code of the selected function, right-click this function and choose **Open Source Code**. You can also use the button in title bar. For more information, please refer to View source code.

5.2.3.5.3Go to callers and callees

Right click any function in this view. The menu list shows all its callers and their addresses. The address of caller is defined to be that of the first instruction in the basic block which calls the selected function. The menu list also displays the information of its target basic blocks, along with addresses of their first instructions and functions name which they belong to. If its target is next basic block, it shows **Fall thru** only. By choosing its callees or callers, you can jump directly to these functions.

The following screen capture below, the selected function has one callee and six callers.



You can verify these relationships by referring to graph column on the right side of view.

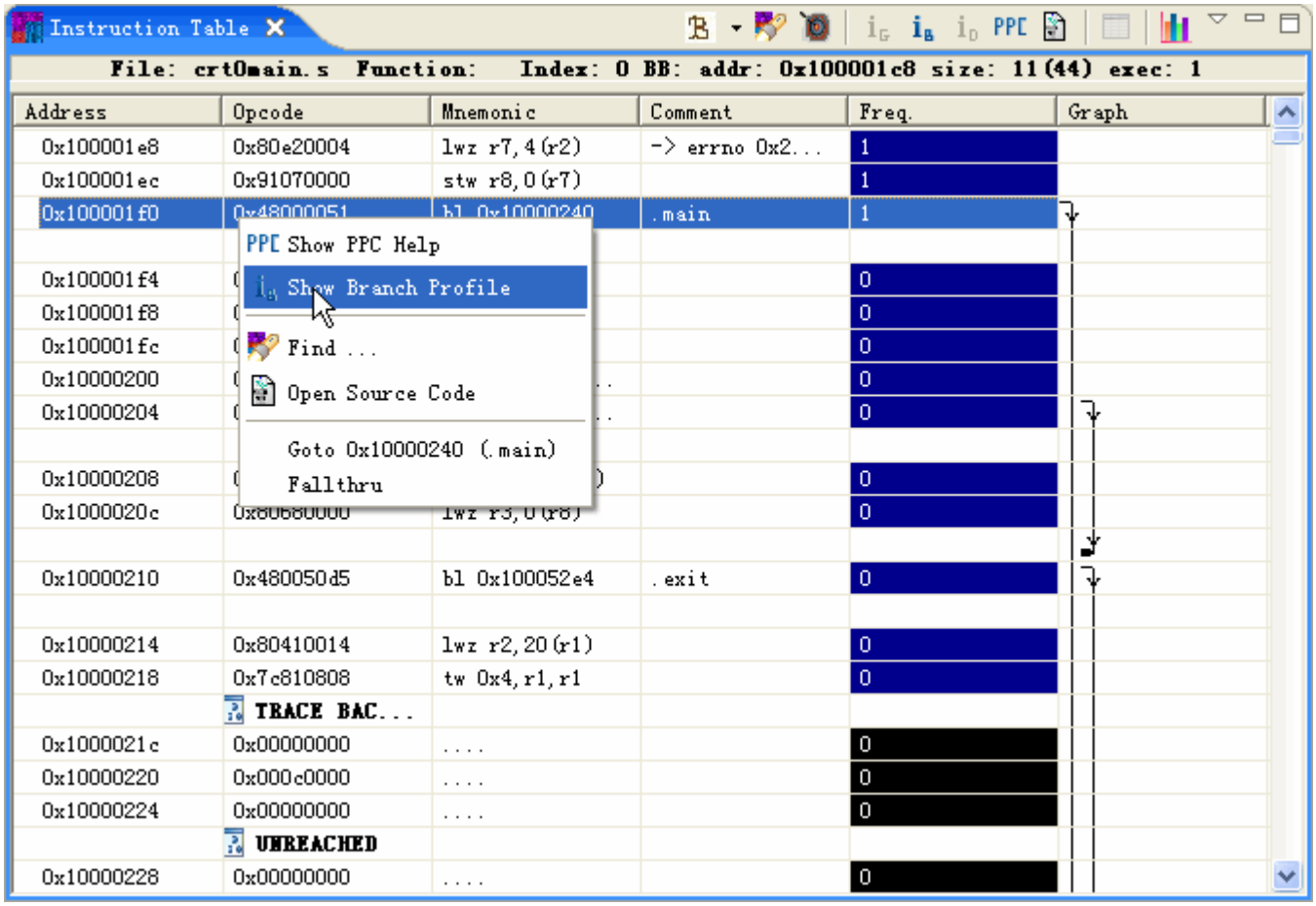
5.2.4Instruction Properties Analysis

The following tasks allow you to analyze instruction properties within Code Analyzer:

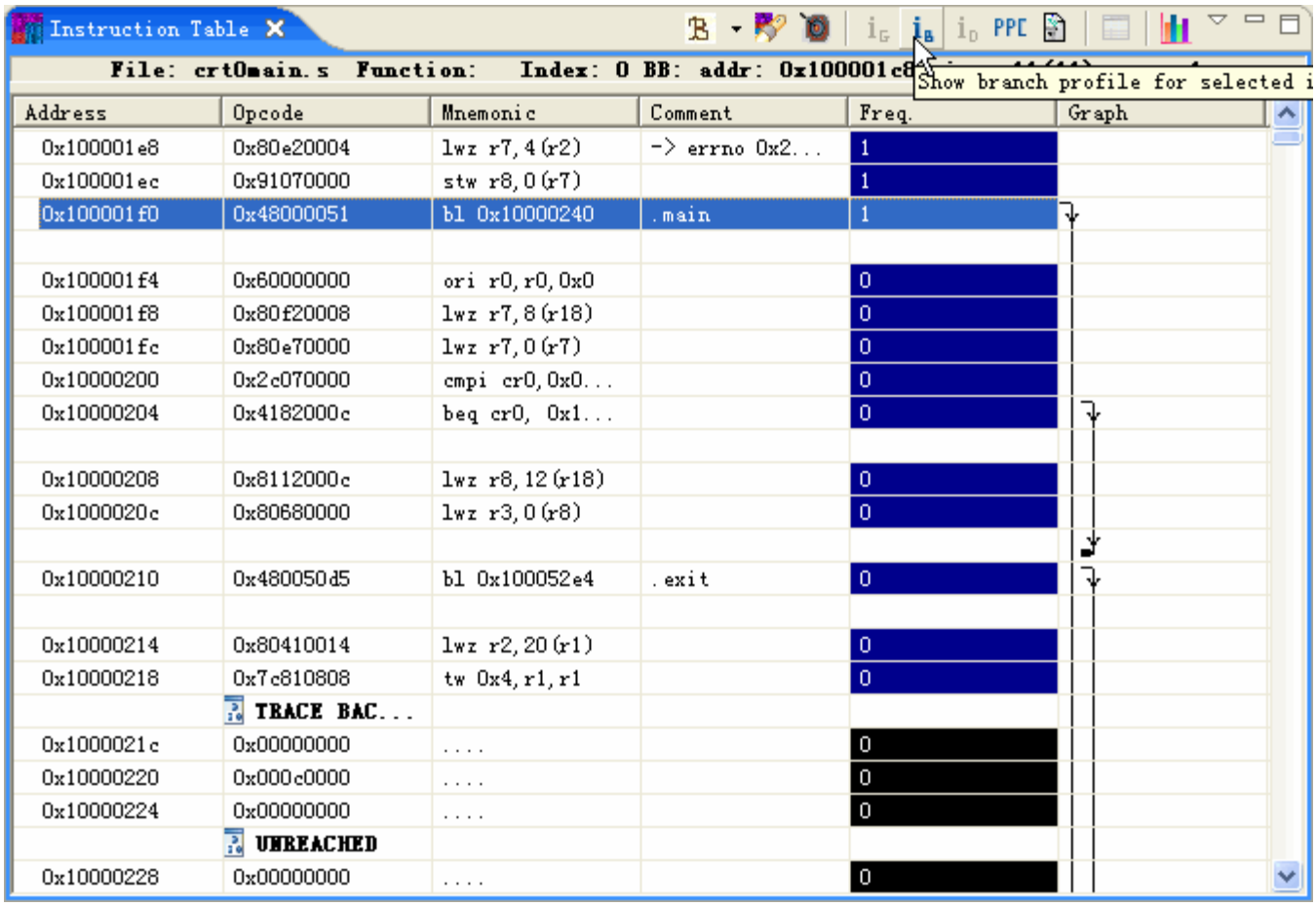
5.2.4.1View Branch Profile

Branch Profile table shows the detailed information of targets of the instruction in the end of an instruction group. This information is available only after loading a profile file.

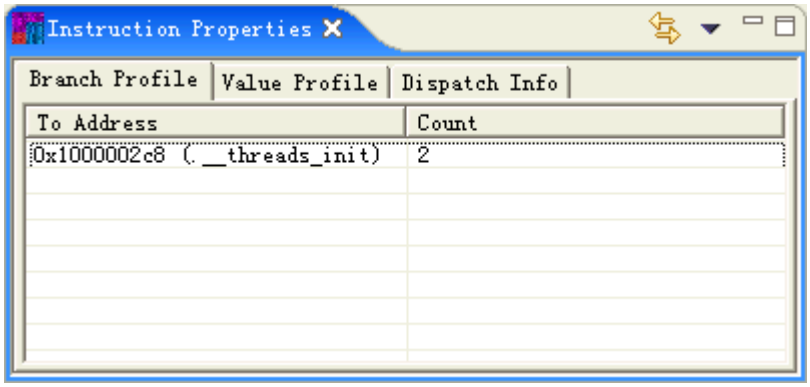
To get the information, please follow these steps: load an executable, add profiling information, be sure to open the Instructions Table, set instructions view in the Instructions Table, select the last instruction of an instruction group and right-click it and then choose **Show Branch Profile**.



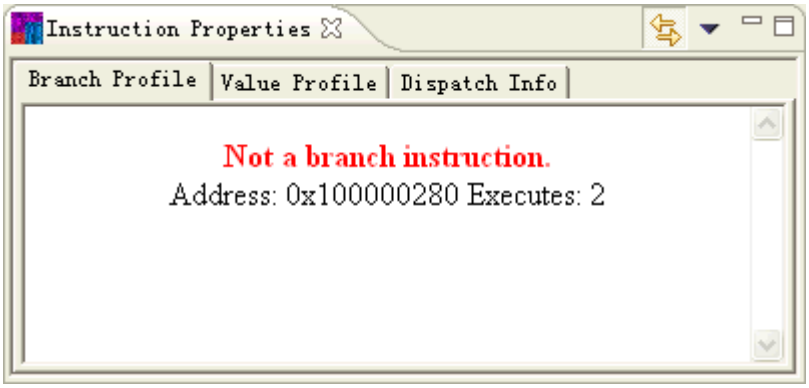
You can also press the button in the Instructions Table's title bar.



Then branch profile tab of instruction properties will appear in the right bottom of the workbench window. It displays the addresses (including function's name) and counts of the target basic blocks of the selected instruction group.



To simultaneously display branch profile information while scrolling along Instructions Table, press in Instruction Properties view's title bar. If you select an instruction within an instruction group, branch profile will show following information:



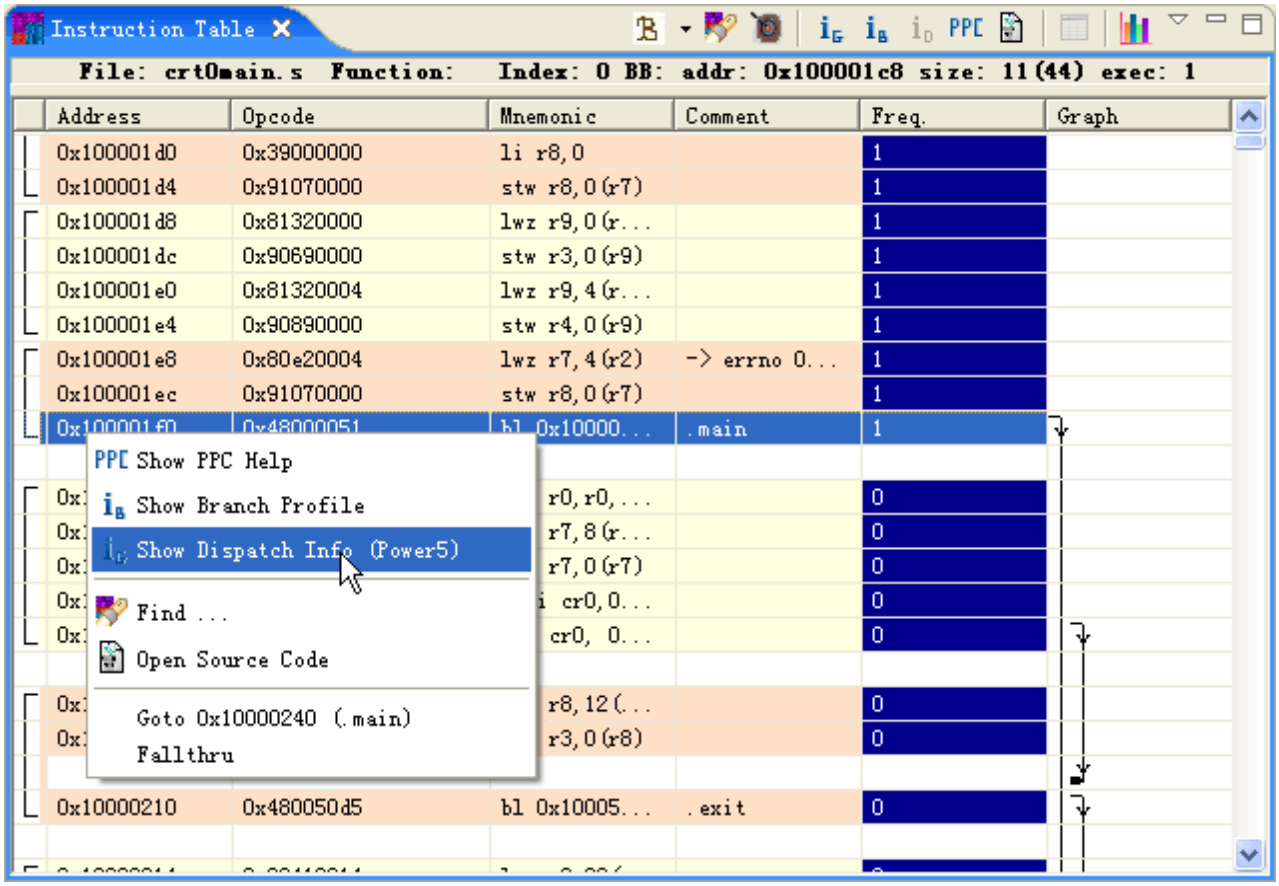
5.2.4.2View Dispatch Information

In Power5 or Power6 architecture, instructions are tracked in groups of one to five instructions rather than as individual instructions. Groups are formed that contain up to five internal instructions, each occupying an internal instruction slot (numbered 0 through 4). Each internal instruction slot in a group feeds separate issue queues for the floating-point units, the branch execution unit, the CR execution unit, the logical CR execution unit, the fixed-point execution units and the load/store execution units. With profile information, CodeAnalyzer can display this information in Dispatch information tab in Instruction Properties view.

To get dispatch information of an executable, please follow these steps: Load an executable, Add the profiling information, open the Instruction Table, Set the instructions view and press  in the Code Analyzer toolbar.

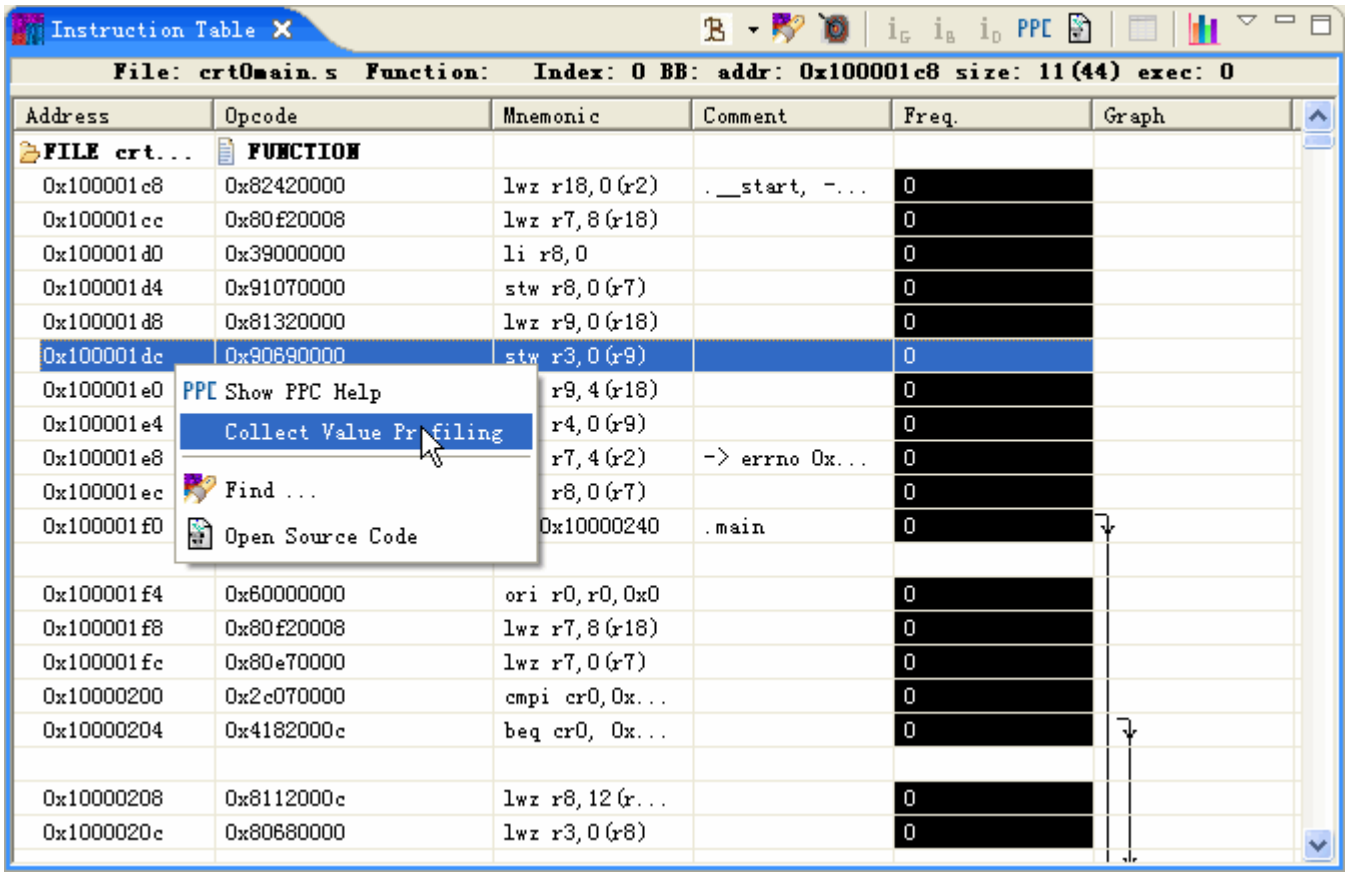
Select the kind of Power architecture your executable is run on in the following wizard.

In the Instruction Table view, select an instruction, right-click and choose Show Dispatch Info. The architecture you have chosen in the previous step will display alongside the menu item.

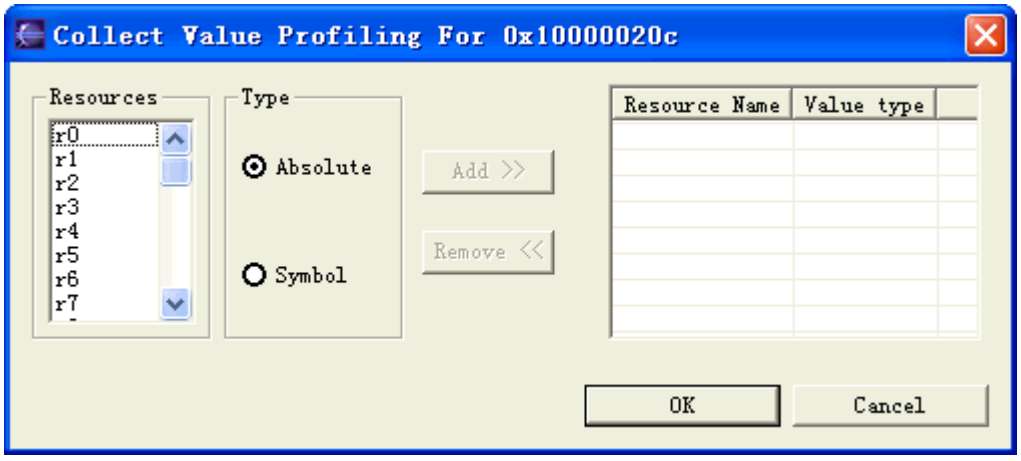


5.2.4.3View Value Profile

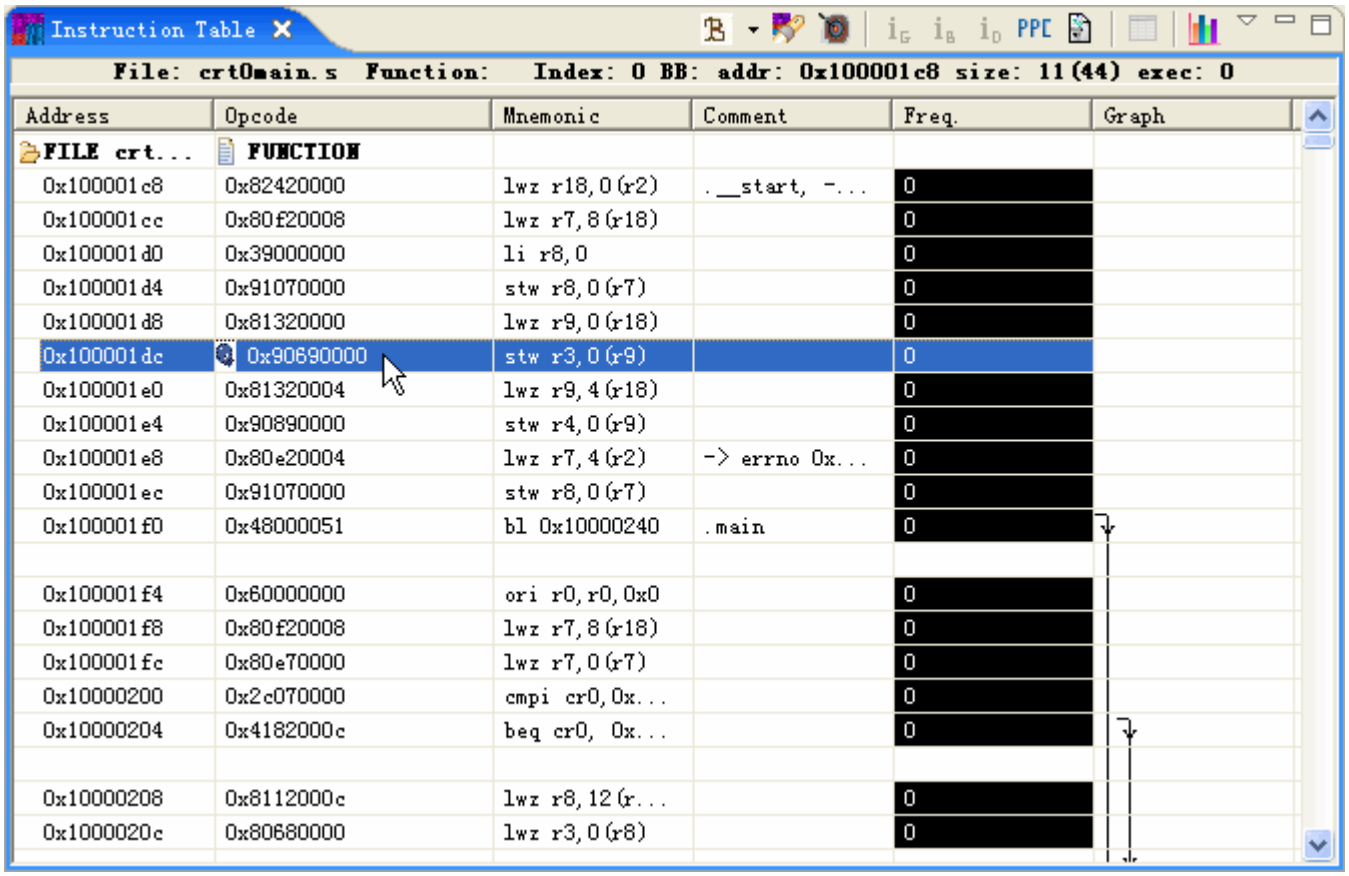
Value Profile is used to show the resources, their values and counts of specific instruction in the loaded executable. To collect these values, please follow these steps: load an executable, open the Instructions Table view and set instructions view and then set to collect resources of some instruction by right-clicking each of them and choose Collect Value Profiling.



Choose resources and their types in the following wizard.



Press **Ok**. The following screen capture shows an icon may appear on the selected instructions after the above steps.



Press in the CodeAnalyzer toolbar to run instrumentation.

Press to write the instrumented file to disk. If you are running on a windows, please make sure to set the output of profile-file value to ./<filename>

Upload the instrumented executable and the profile file you have created to an AIX machine. Make sure to put them under the same directory.

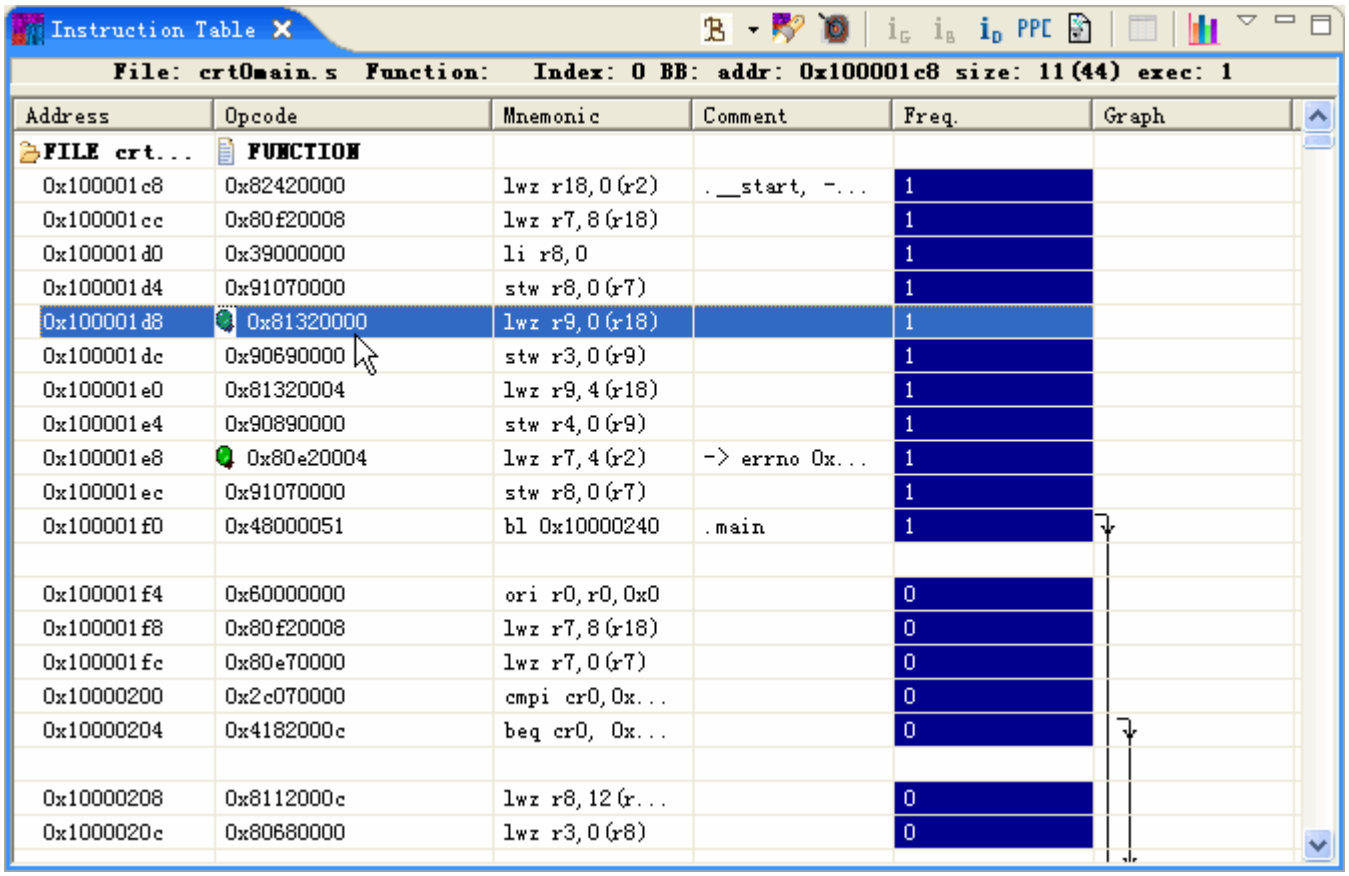
Run the instrumented executable with some training data. It will write information of collected value to your original profile file. You can get relevant training data from SPEC2000.

Copy the profile file created back to the windows.

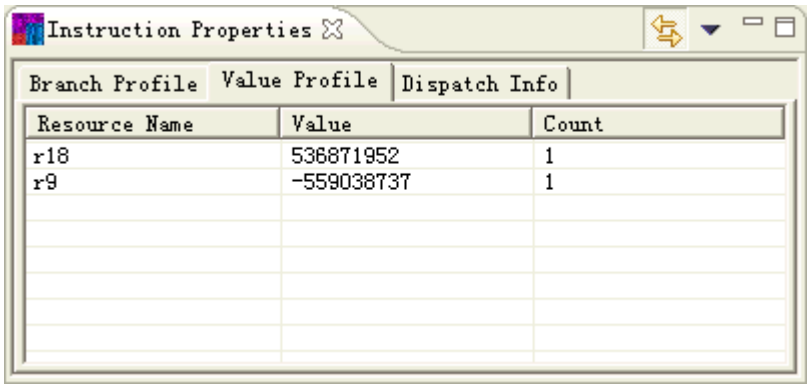
Load the original executable in CodeAnalyzer again.

Add profile file you've created in --profile--file.

The following screen capture shows that the original grey icon may change to green icon in the front of the selected instruction after doing the above steps.

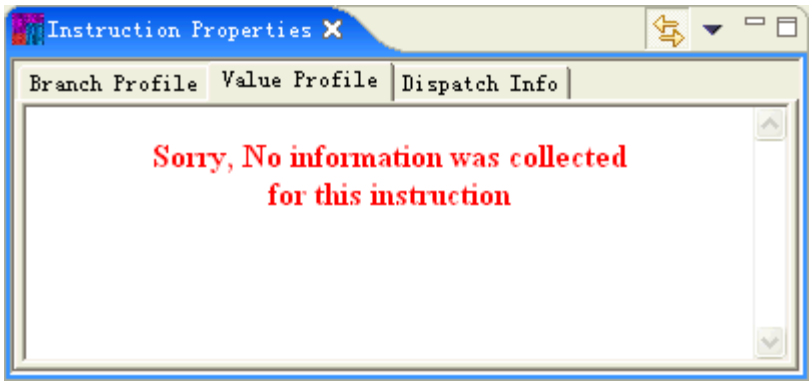


To view the value of collected resources, right-click the marked instruction and choose **Show Value Profile**. The resources value will be displayed in Value Profile table. You can also press in Instructions Table view's title bar.



To simultaneously display value profile information while scrolling along Instructions Table, press

in Instruction Properties view's title bar. If you select an instruction without green icon, value profile will show as follows:



5.2.4.4Collect Comments

CodeAnalyzer can display comments generated by FDPR-Pro engine. Different type of files have different comments for collection. So far there are three sources of comments: Power 5, Power 6 and general comments. All the general comments are dependent on profile information. To view there comments, you need to collect comments first. Please follow the steps below:

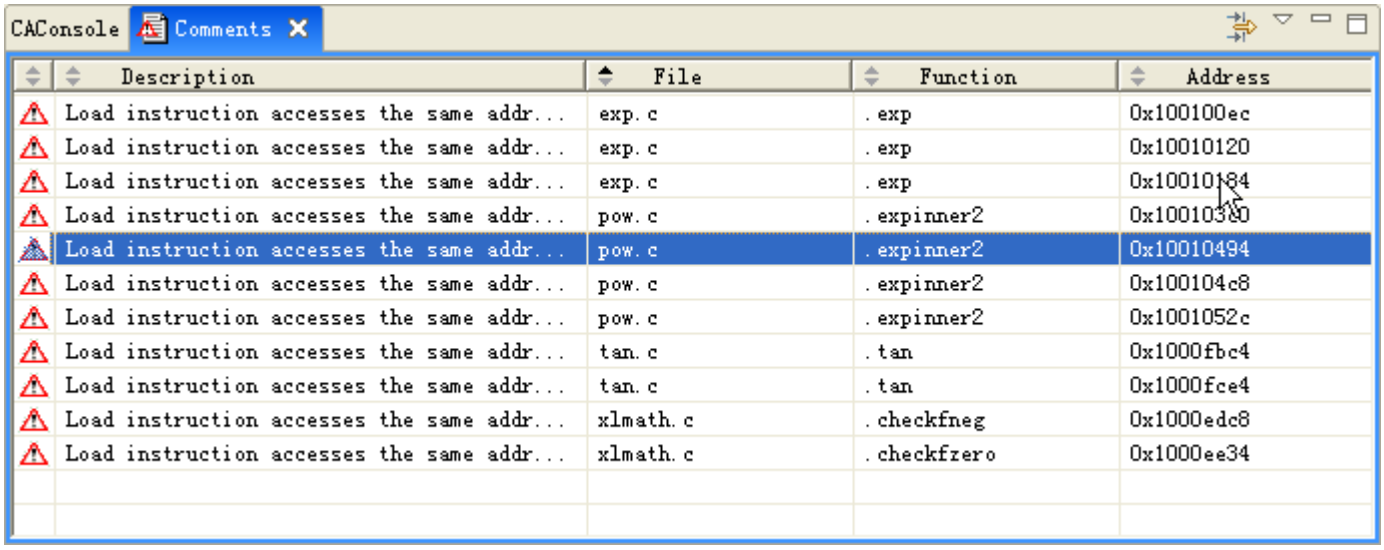
Load an executable

Add profile information of this executable

Be sure to open Instructions Table

Select **File - > Code Analyzer - > Collect Hazard Info** or press button to choose type of comments to collect

Press button to open Comments View



Press button or in tool bar to navigate each comment in **Instruction table**

Press button to display the currently collected comments statistics

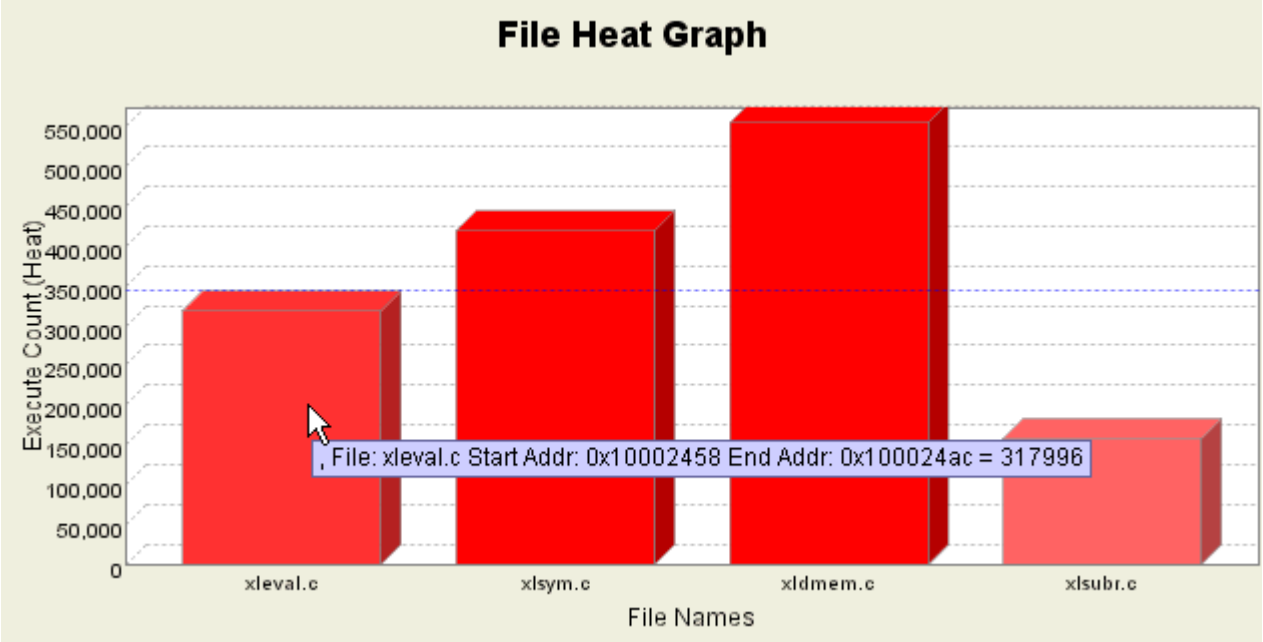
Please notice that if you have restricted grouping to some architecture, the comments view may remove inappropriate comments from it.

5.2.5Statistic Analysis

Code Analyzer provides users with a graphical display of statistics gathered on the loaded executable based on different degrees of granularity. There are three perspectives of analysis: file, function and instruction mix. They are shown in the form of tab in Statistics view. All the graphs are drawn to scale in cylinder. Each column is colored according to its frequency heat. The Statistics view normally displays top (hottest) files, functions or instructions. In the upper level of each tab, there is filter button. Only items that pass the average threshold you set in filter value will be displayed. Therefore, if you enter 0% and press **refresh**, the view will show all the files. And if you enter 100% and press **refresh**, a single column will be displayed, representing the hottest execution unit.

To open these views, you need to load an executable and add profile information first. Then press the corresponding buttons in Code Analyzer's toolbar. You can also open it from **File -> Code Analyzer -> Statistics**.

When your cursor stops in a column in the graph for a while, there is reference information like this:

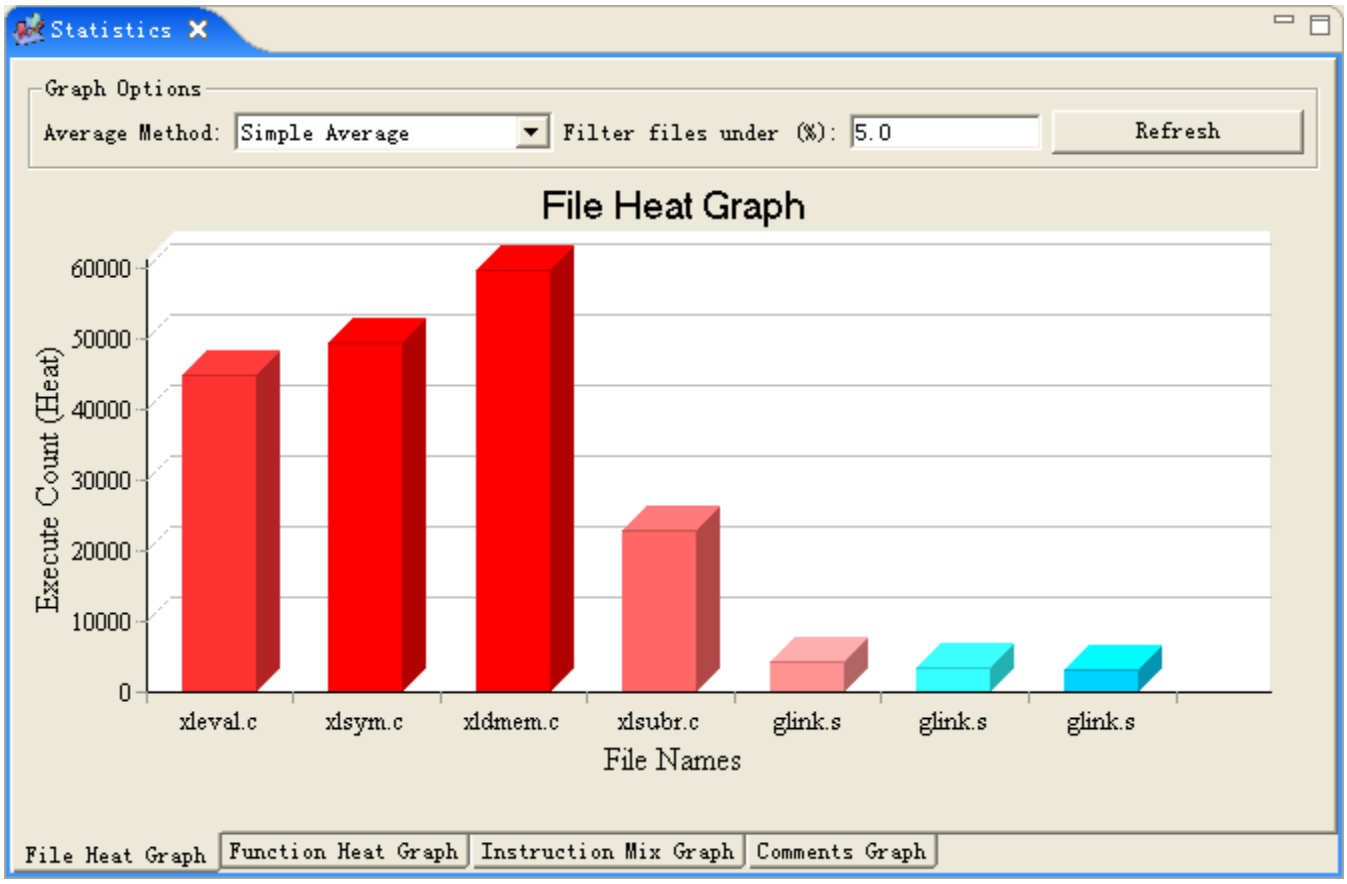


5.2.5.1View File Heat Graph

To open file heat graph, press button  in CodeAnalyzer toolbar or choose **File -> CodeAnalyzer -> Statistics -> Files Heat**. You can also select the tab within Statistics view .

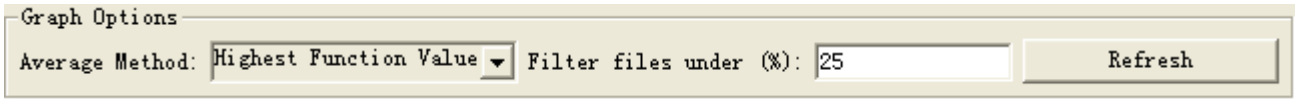
After that, load an executable and add necessary profile information.

The following screen capture shows a typical file heat graph of loaded executable.

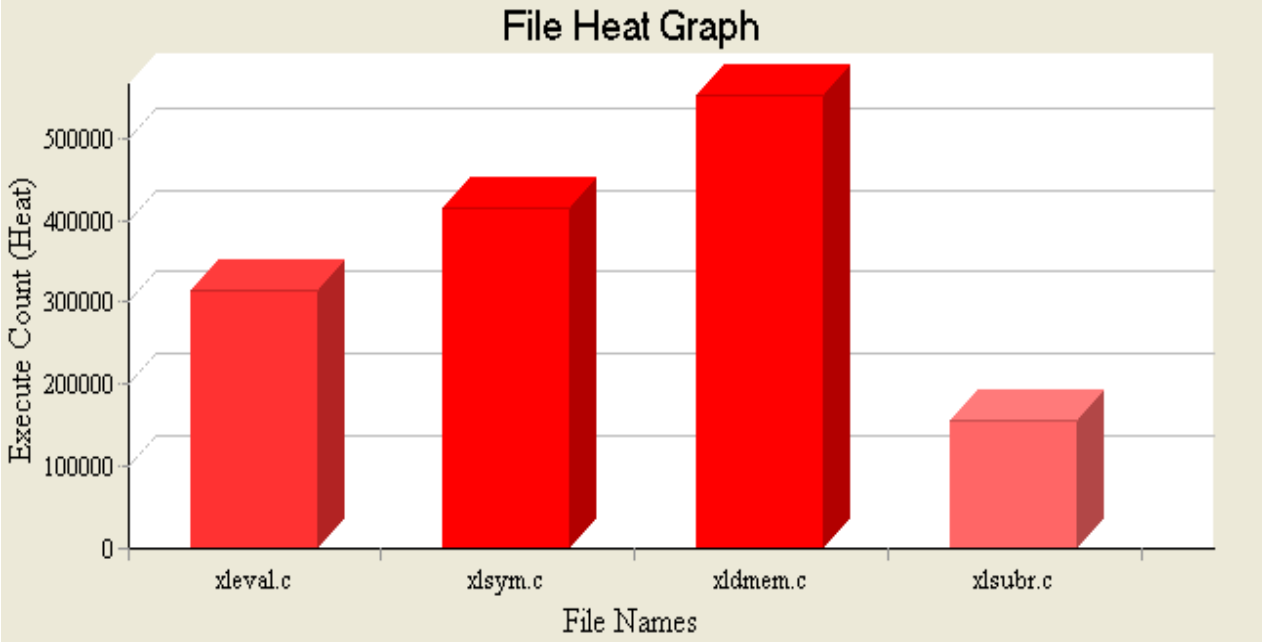


The x-ordinate shows the names of files in the loaded executable. The y-ordinate denotes execute count distribution. Each column of the graph refers to a file in the loaded executable. Their color shows how frequently they are called. You can set graph options to customize the view. There are three methods in the average method box: simple average, weighted average and highest function value. By inputting the minimum percentage in the latter box and press **Refresh**, you can get the count distribution of those files whose percentages calculated by average method are above this value.

For example, if you use the highest function value as average method and try to filter 25% files, the graph options should be set as follows:



Press **Refresh**. Then the files whose highest function value percentages are at least 25% of the maximum value will be listed in the graph as follows:

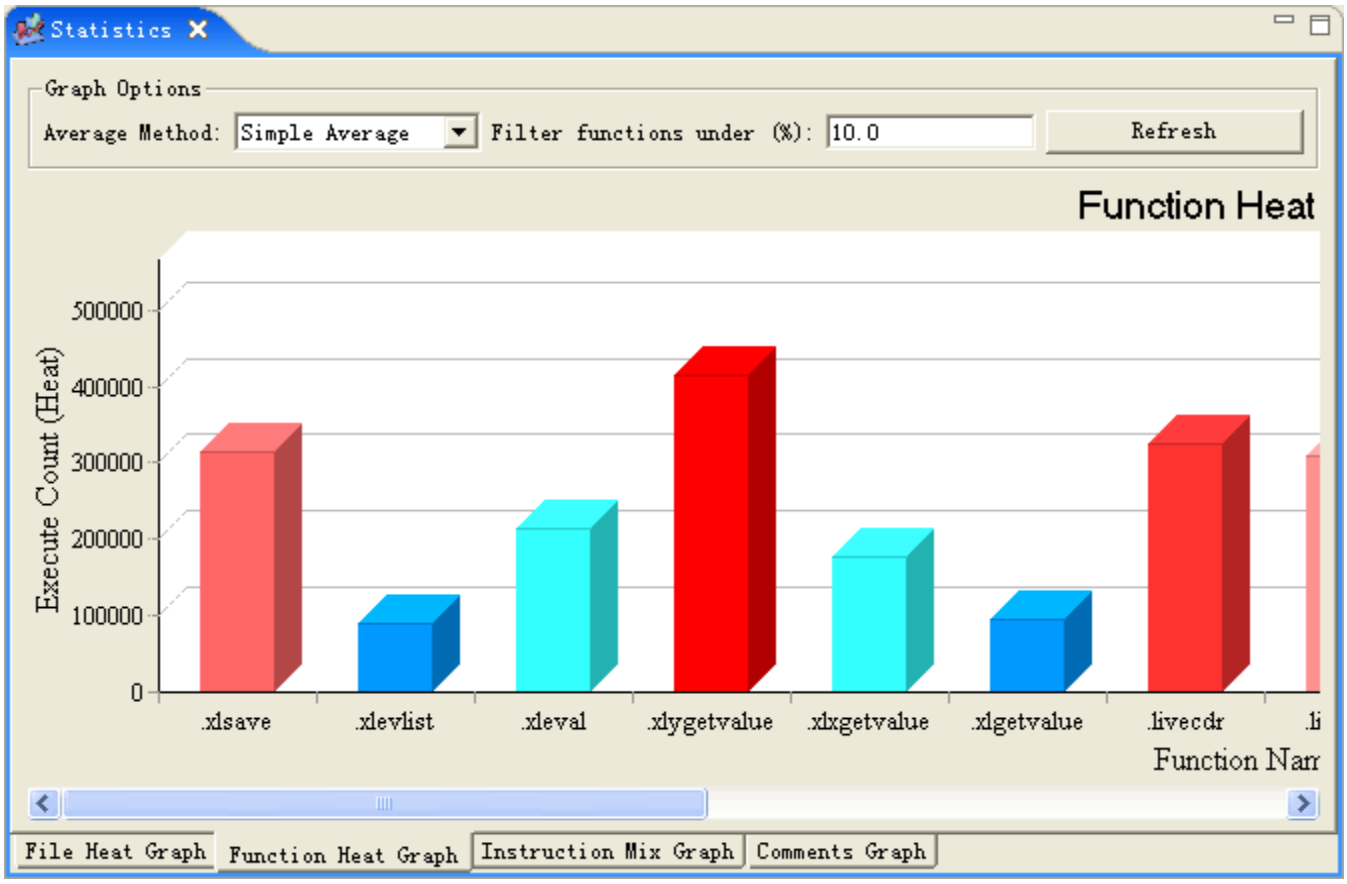


5.2.5.2View Function Heat Graph

To get function heat graph, you can press button in CodeAnalyzer toolbar or choose **File -> CodeAnalyzer -> Statistics -> Functions Heat** . You can also select the tab in Statistics view .

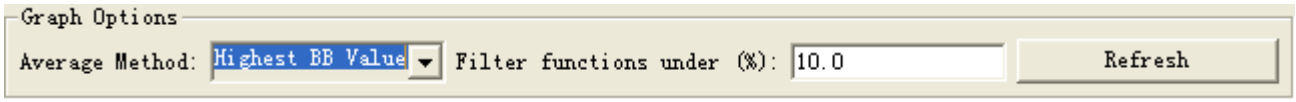
After that, load an executable and add necessary profile information.

The following screen capture shows a typical function heat graph of loaded executable.

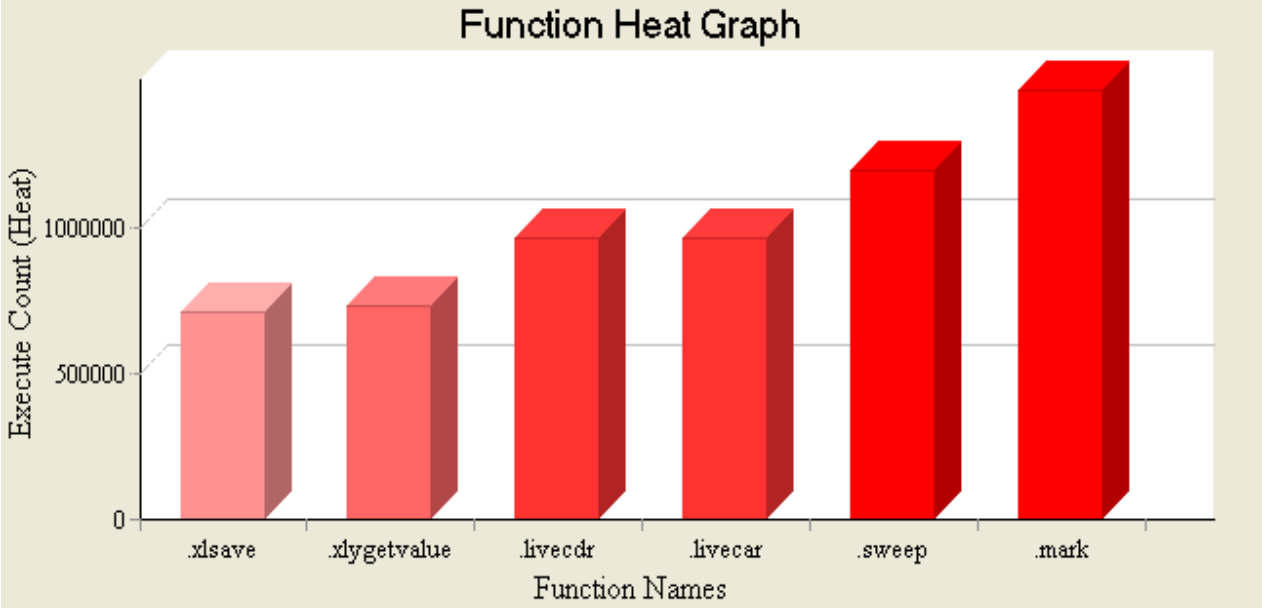


The x-ordinate shows the names of functions in the loaded executable. The y-ordinate denotes execute count distribution. Each column of the graph refers to a function in the loaded executable. Their color shows how frequently they are called. You can set graph options to customize the view. There are four average methods: simple average, weighted average, highest BB value and prolog value. By inputting the minimum percentage in the latter box and press **Refresh**, you can get the count distribution of those functions whose percentages calculated by average method are above this value.

For example, if you use the highest BB value as average method and try to filter 25% files, the graph options should be set as follows:



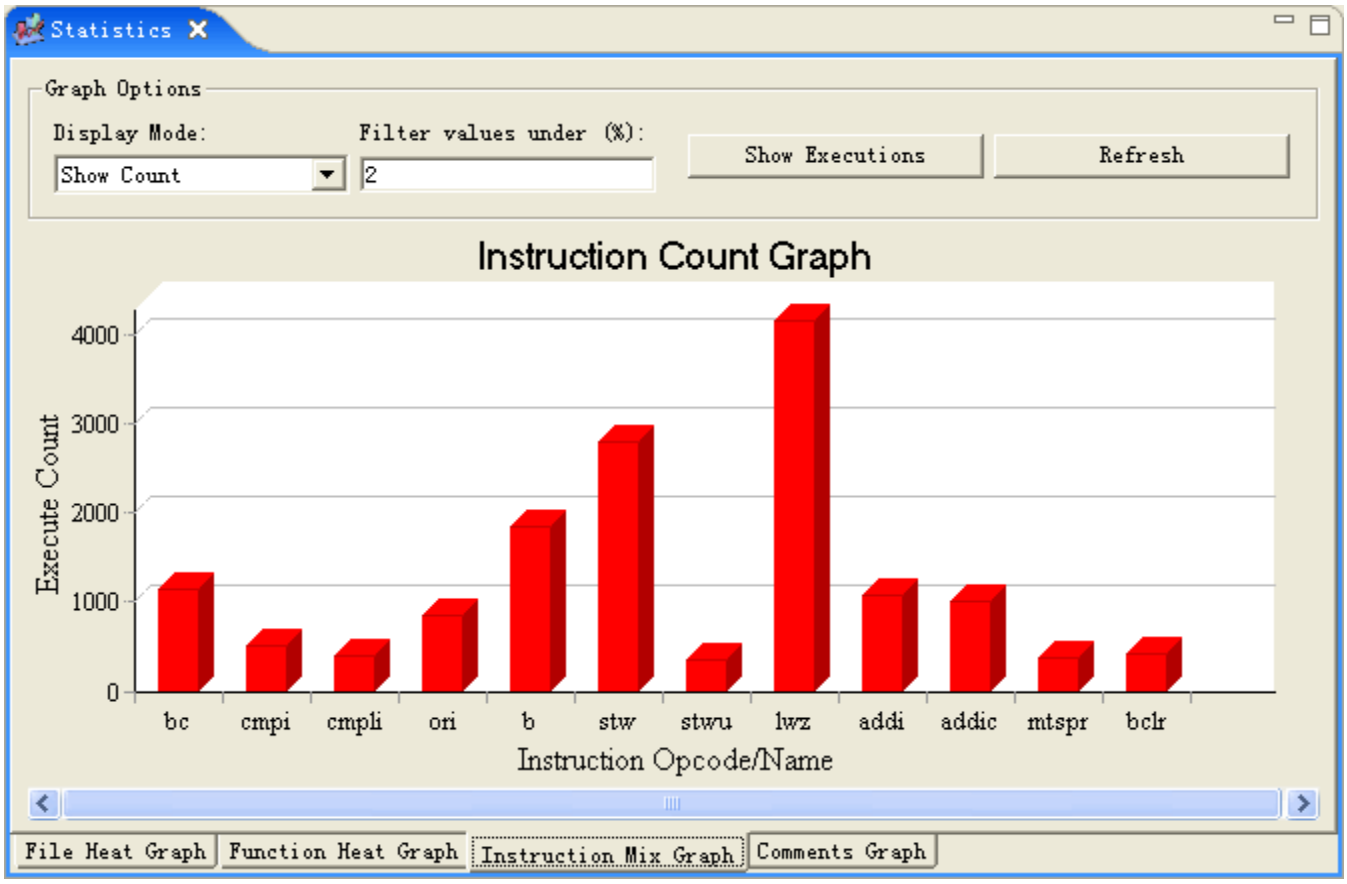
Press **Refresh**. Then the functions whose highest BB value percentages are at least 25% of the maximum value will be listed in the graph as follows:



5.2.5.3View Instruction Graph

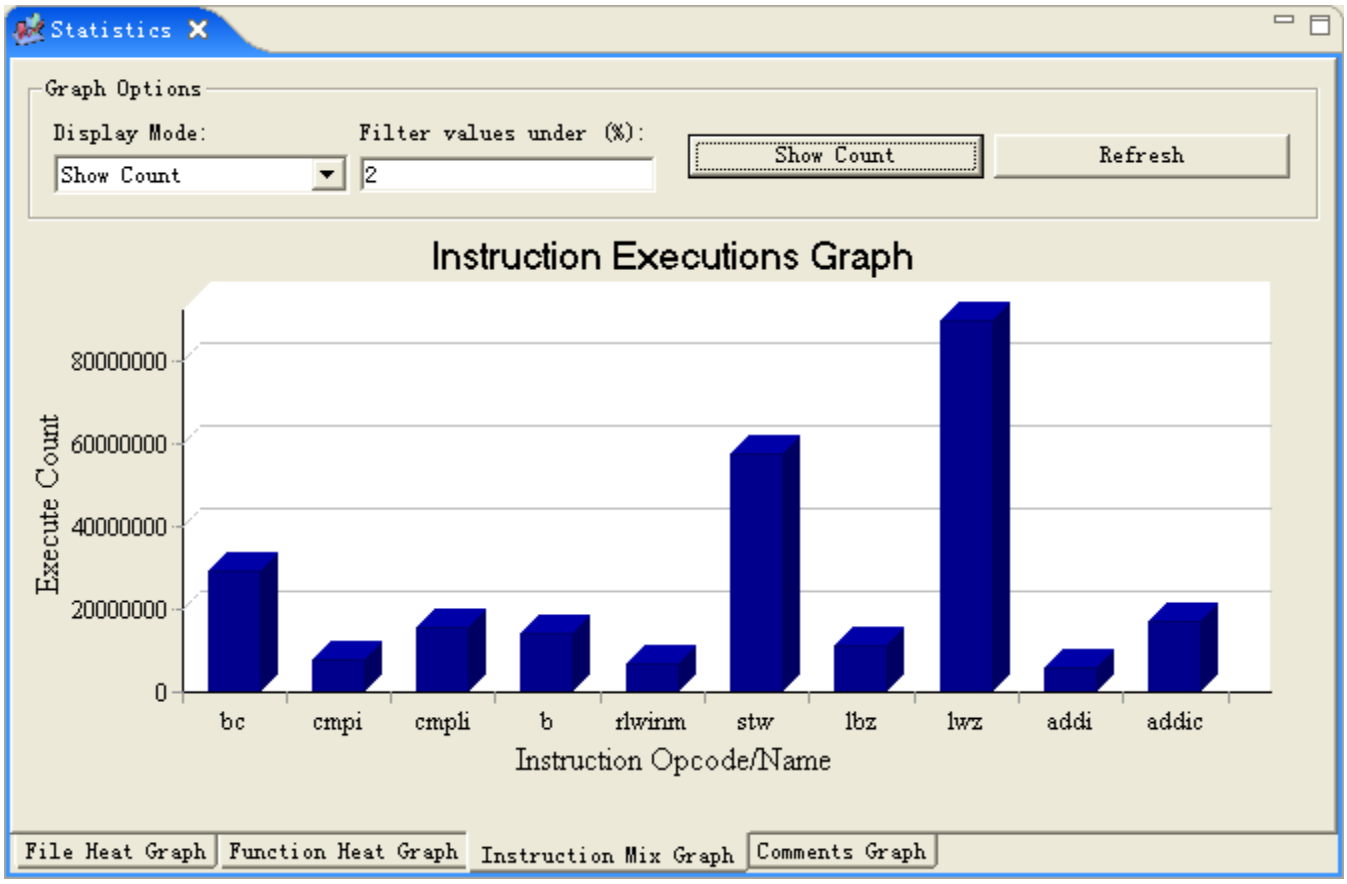
To open instruction graph, press button in CodeAnalyzer toolbar or choose **File -> CodeAnalyzer -> Statistics -> Instruction Mix**. You can also select the tab in Statistics view . There are two modes of instruction mix graph: count and percentage. You can press the button **Show Executions** or **Show Count** to switch between them.

The following screen capture shows a typical instruction count graph of loaded executable.



The x-ordinate shows the names of instructions in the loaded executable. The y-ordinate denotes count distribution of instructions. Each column of the graph refers to an instruction. Their color shows how frequently these instructions appear in the loaded executable.

The following screen capture shows a typical instruction executions graph of loaded executable.

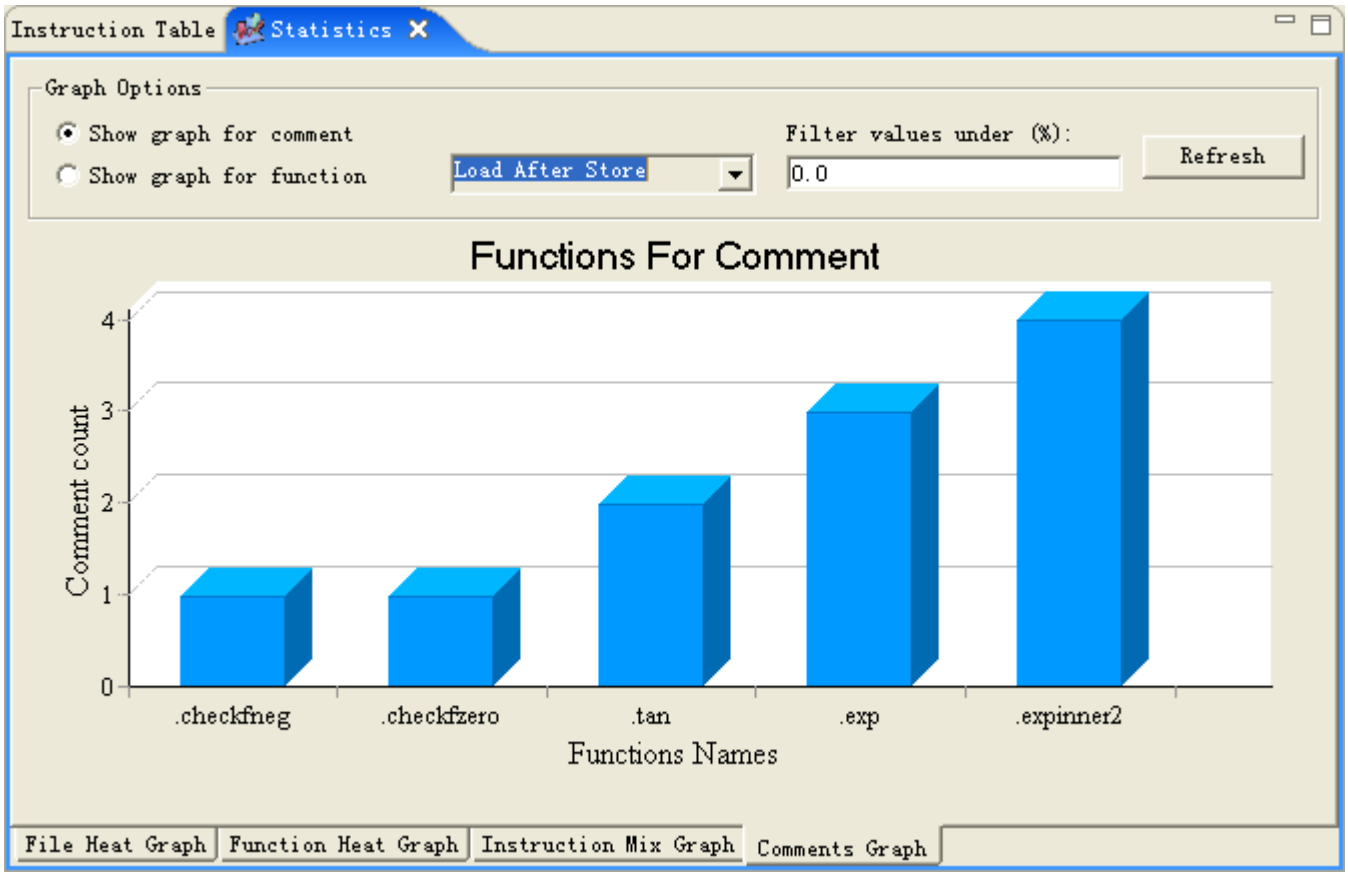


The x-ordinate shows the names of instructions in the loaded executable. The y-ordinate denotes executions distribution. Each column of the graph refers to an instruction. Their color shows how frequently these instructions are executed in the loaded executable.

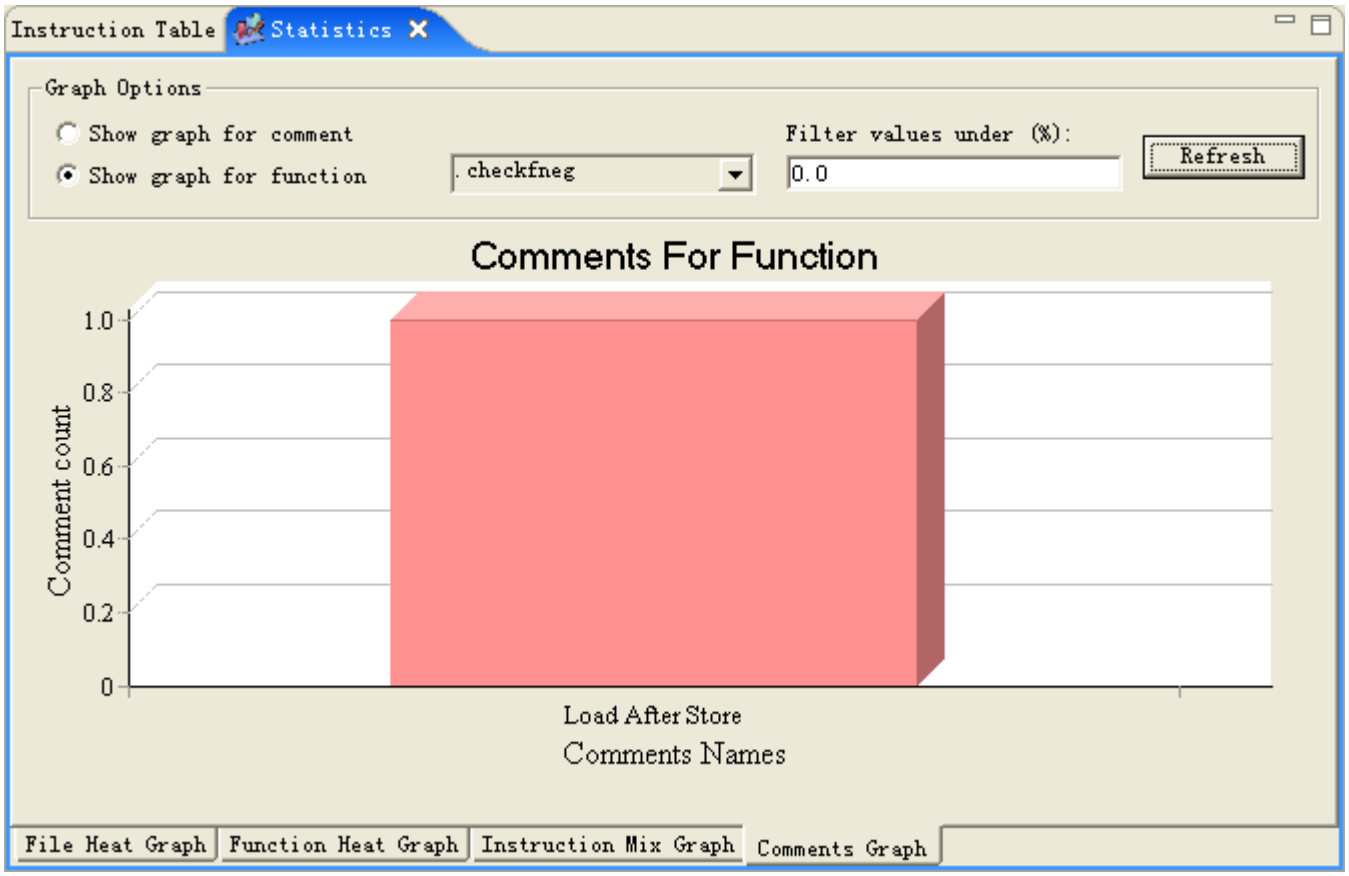
5.2.5.4View Comments Graph

To get comments graph, be sure to collect comments first.

Then press button in toolbar or choose **File -> CodeAnalyzer -> Statistics -> Comments** . You can also select its tab in Statistics view directly. There are two options of graph: **show graph for comment** and **show graph for function**. In graph for comments, you can select type of comments to display.



In the above graph, the x-ordinate shows the names of functions that have comments of **Load After Store** in power 5. The y-ordinate denotes number of this comment for each function. Each column of the graph refers to a function.



In graph for function, you can select any of functions which have comments and display the number of different comments it contains. In the above graph, the x-ordinate shows the name of comments we try to collect in the first step. The y-ordinate denotes number of this comments in **.checkfneg** function. You can filter the value of columns by pressing **Refresh**.

5.3Pipeline Analyzer

Pipeline Analyzer is a port of the IBM Performance Simulator for Linux on POWER™, another alphaWorks technology. Pipeline joins the VPA toolkit to provide VPA users with the means of examining how code is executed on various IBM POWER processors. Pipeline Analyzer displays the pipeline execution of instruction traces generated by a POWER series processor. It does so by providing a scroll view and a resource view of the instruction execution.

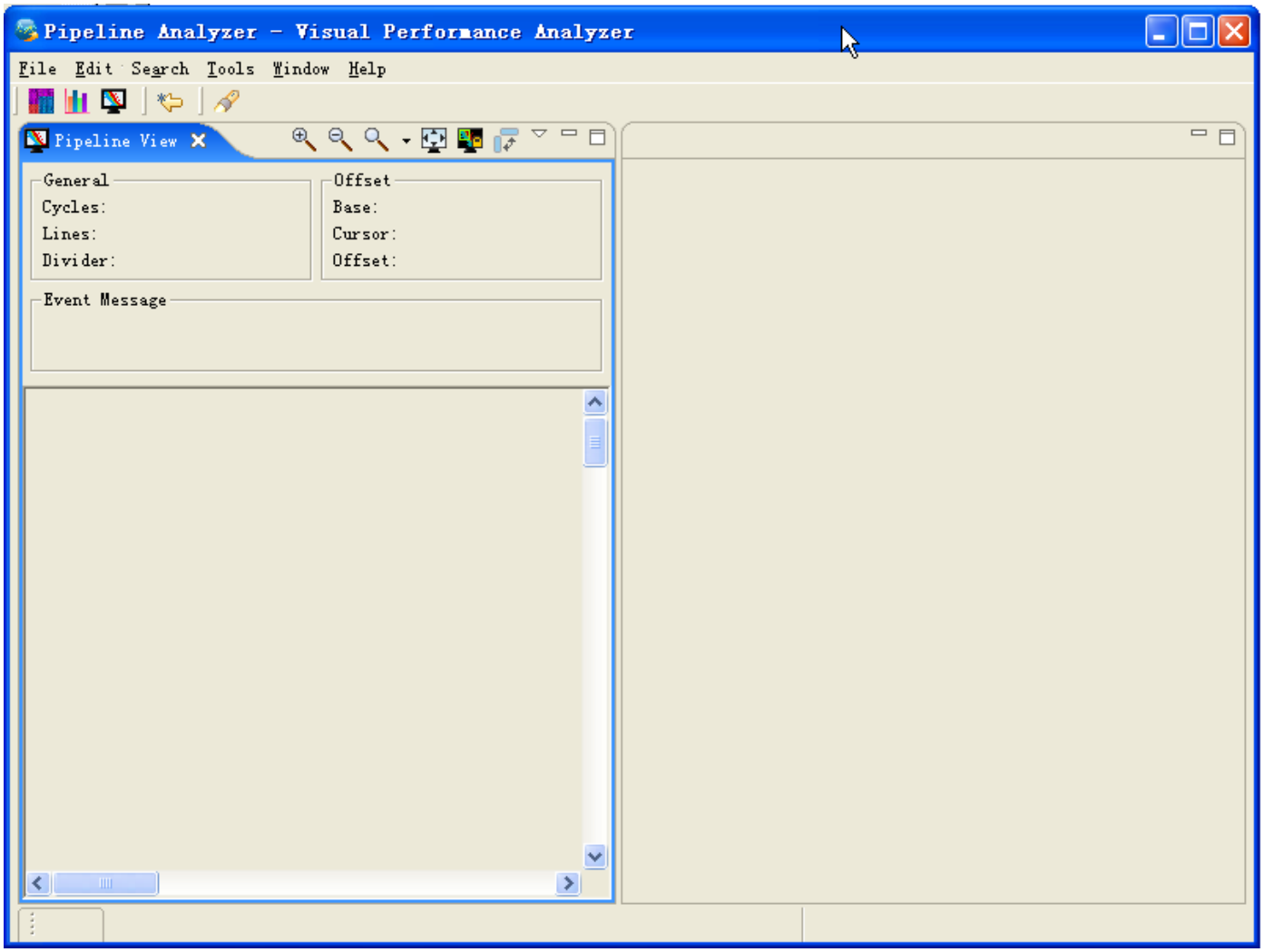
You can also find the Pipeline Analyzer User Guide from within VPA. Select Help - Help Contents within VPA. To get context sensitive help, press F1 for Windows and AIX or press Ctrl+F1 for Linux.

5.3.1Load an existing pipeline file

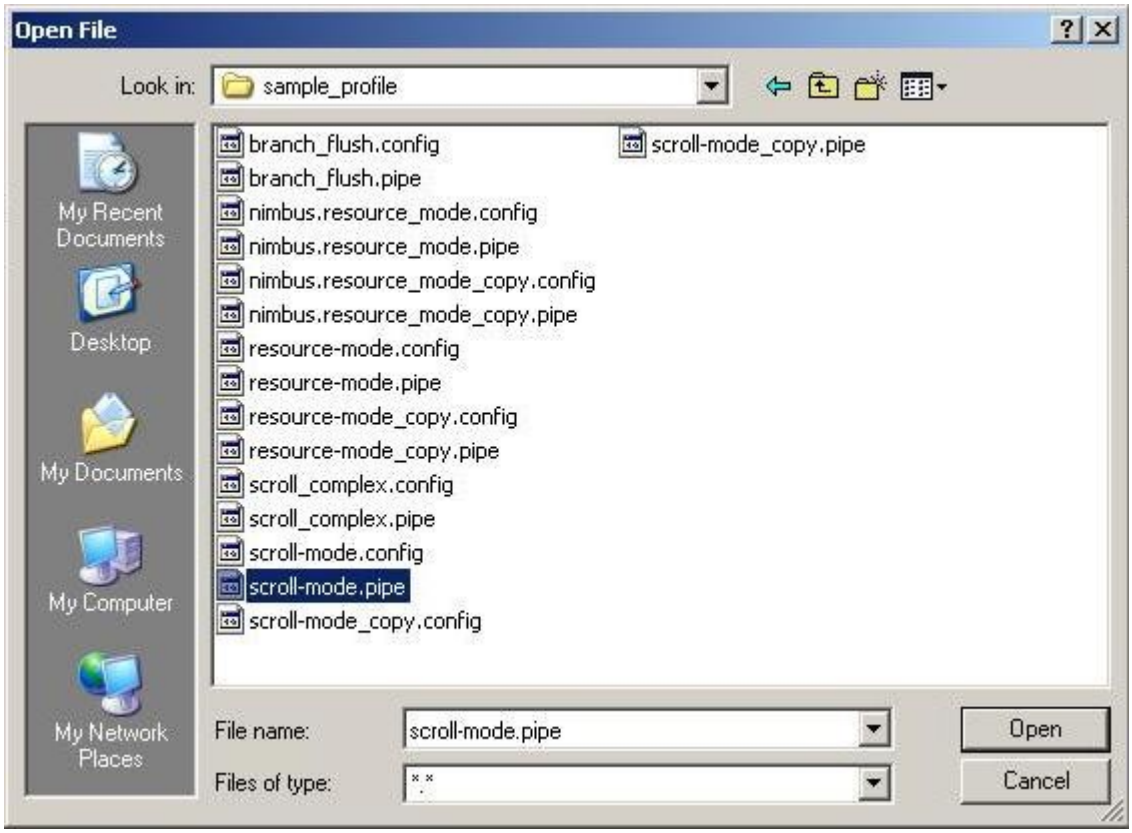
The IBM Performance Simulator for Linux on POWER™ project has directions for capturing an instruction trace and generating Pipeline data files.

Once you have made a run and generated a .pipe and .config file you can use the Pipeline Analyzer to look at them. When you start Visual Performance Analyzer, the default perspective is Profile Analyzer Perspective. To open Pipeline Analyzer, choose **Window -> Open Perspective -> Other -> Pipeline Analyzer** .

The following screen capture shows the initial layout of Pipeline Analyzer.

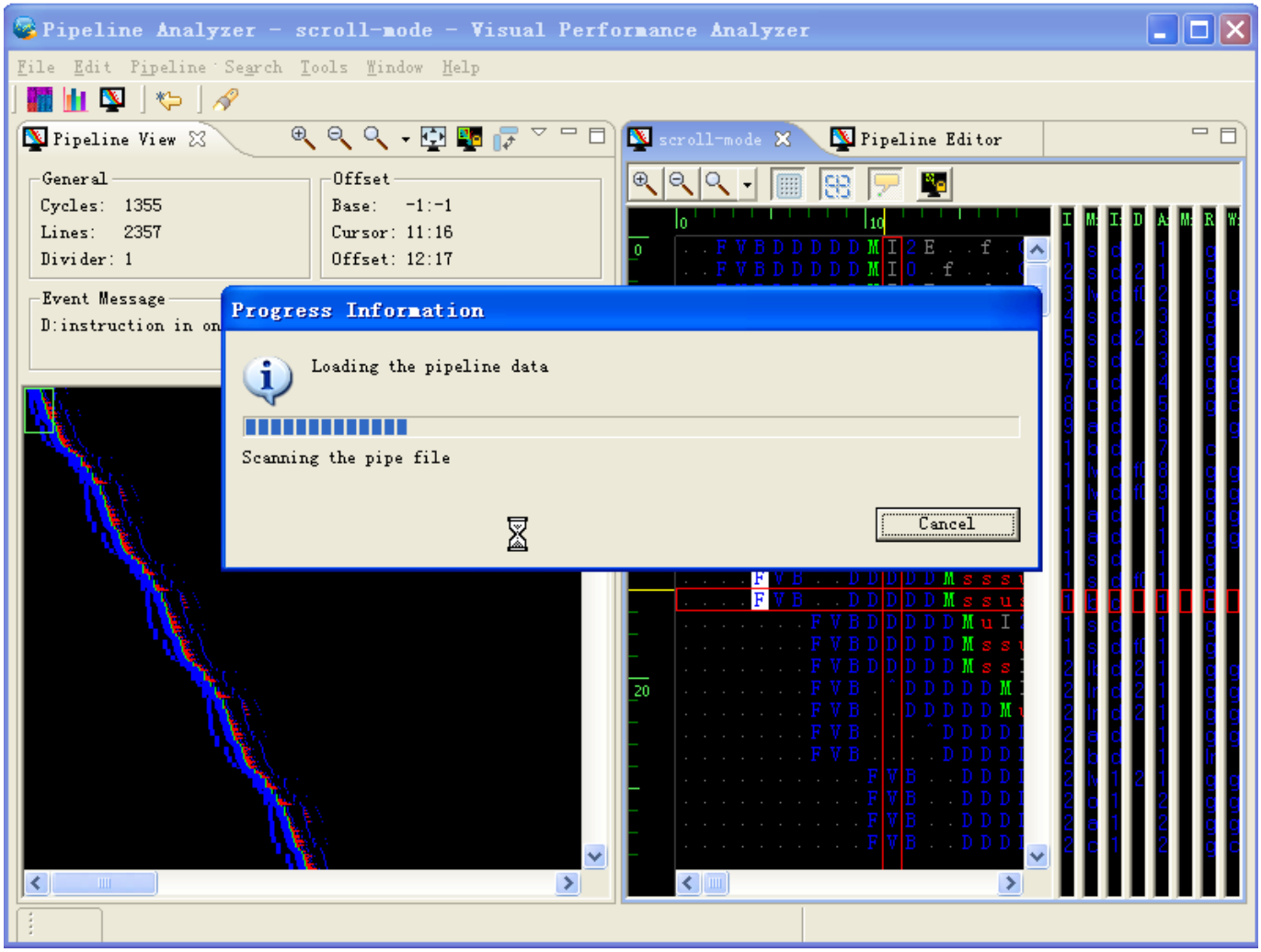


Choose **File -> Open File**_ , and in Open File dialog select .pipe file with scroll mode information for inspection.

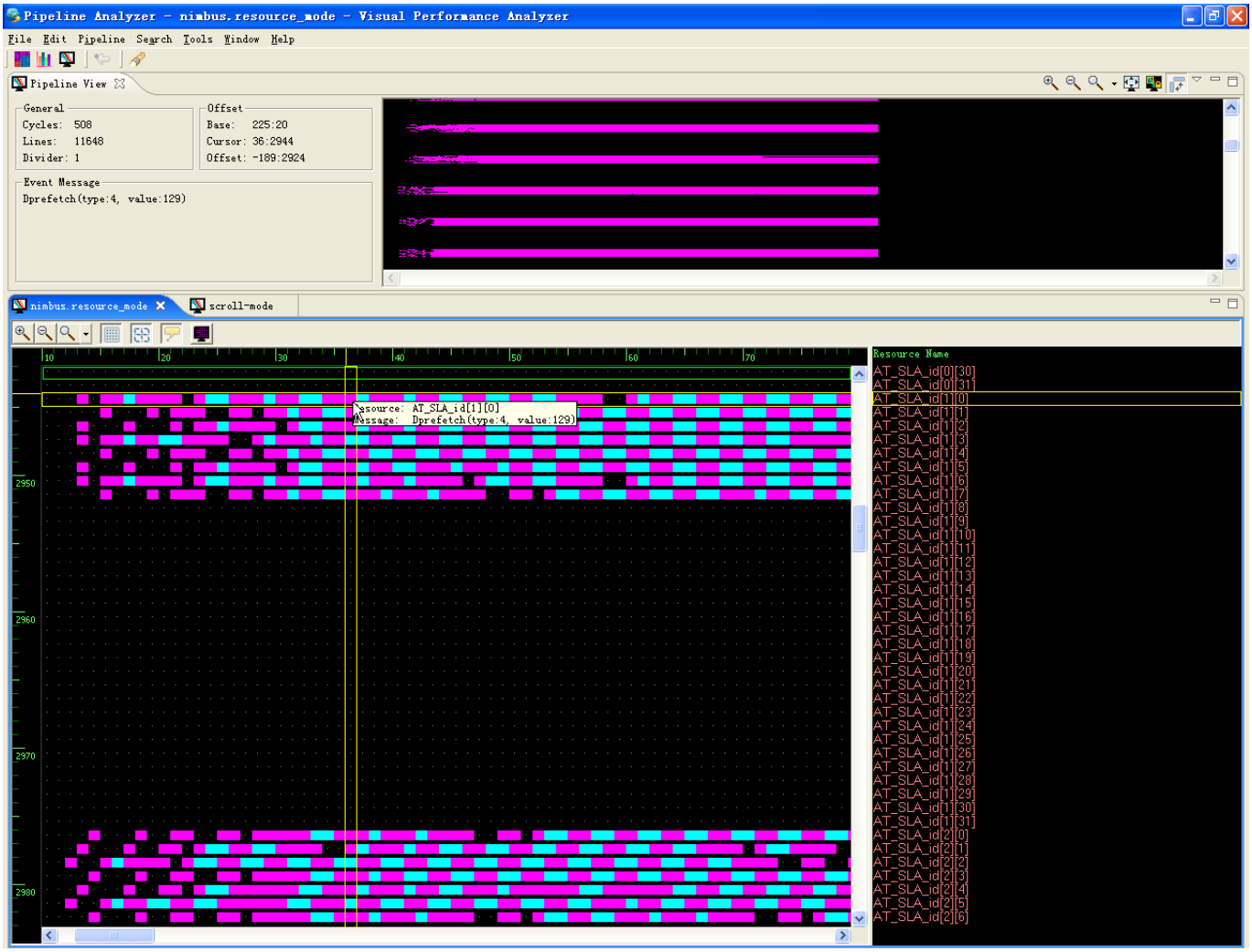


Please note if your .config file has different name as .pipe file, a second dialog for corresponding .config file will turn up for you to choose.

Next, the Pipeline Analyzer Perspective loads the data of .pipe file. A scroll editor is opened automatically. The general information of this .pipe file is displayed in the panel of Pipe View. The following screen capture shows the data loading of this file.



To open .pipe file with resource mode information, choose **File -> Open File**_. A resource editor is opened as follows:



5.3.2Navigating the scroll pipe view

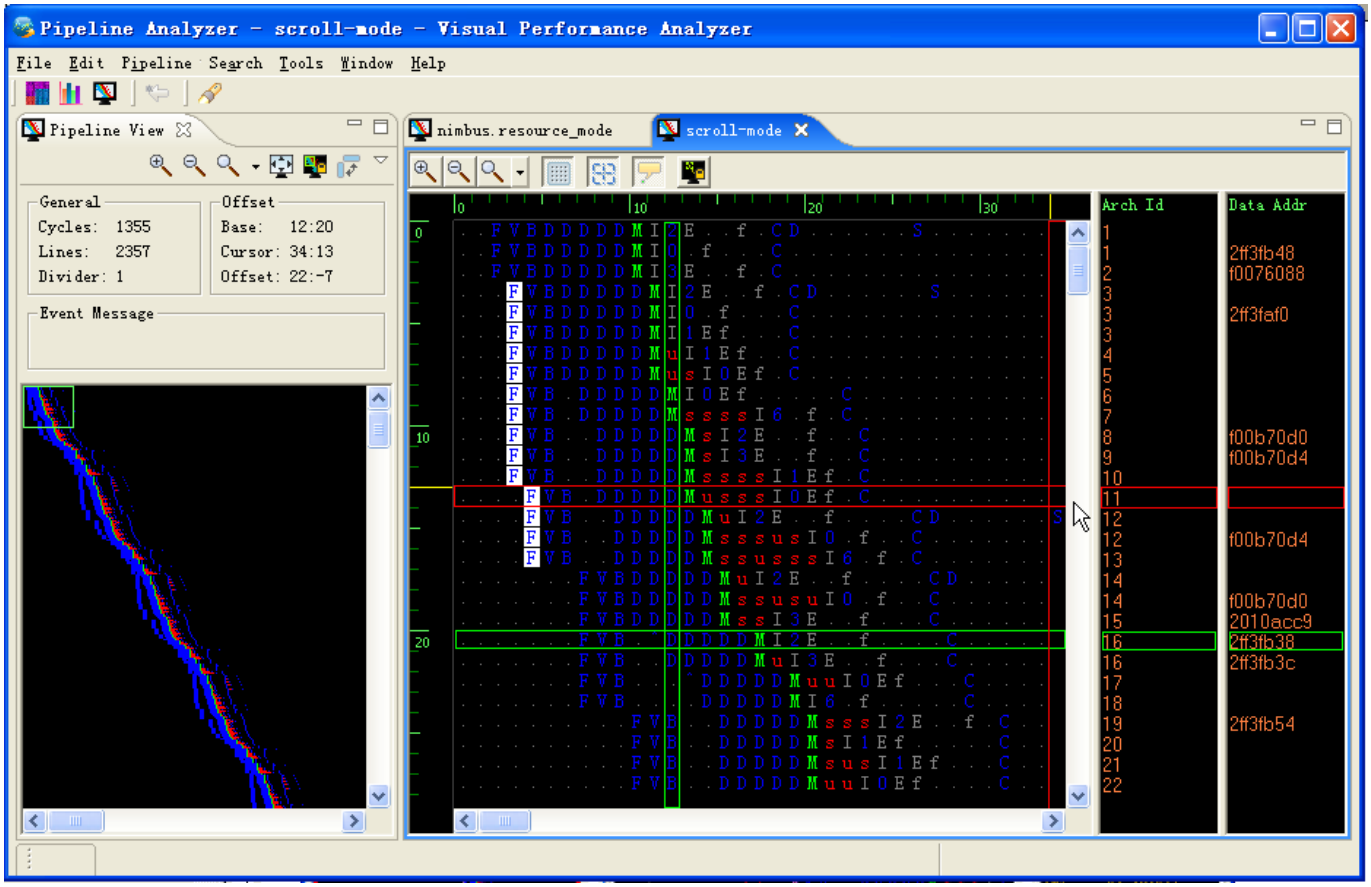
Each time Pipeline Analyzer Perspective is opened, a pipe view turns up in the left part of perspective. It shows the detailed information of the currently active editor.

To open Pipe View manually, choose Window -> Show View -> Pipeline Category -> Pipe View. Note every perspective contains only one pipe view.

To view pipeline file containing scroll mode information, select File -> Open File and choose corresponding file. A scroll editor opens in the Pipeline Analyzer Editor and Pipe View display its information at the same time.

5.3.2.1Using the overview graph

In the overview graph, the green box indicates the boundary of data displayed in the currently active scroll editor. To display data elsewhere, click its location in the graph. You will see that the green box moves to where you just click and scroll editor displays the data in detail.

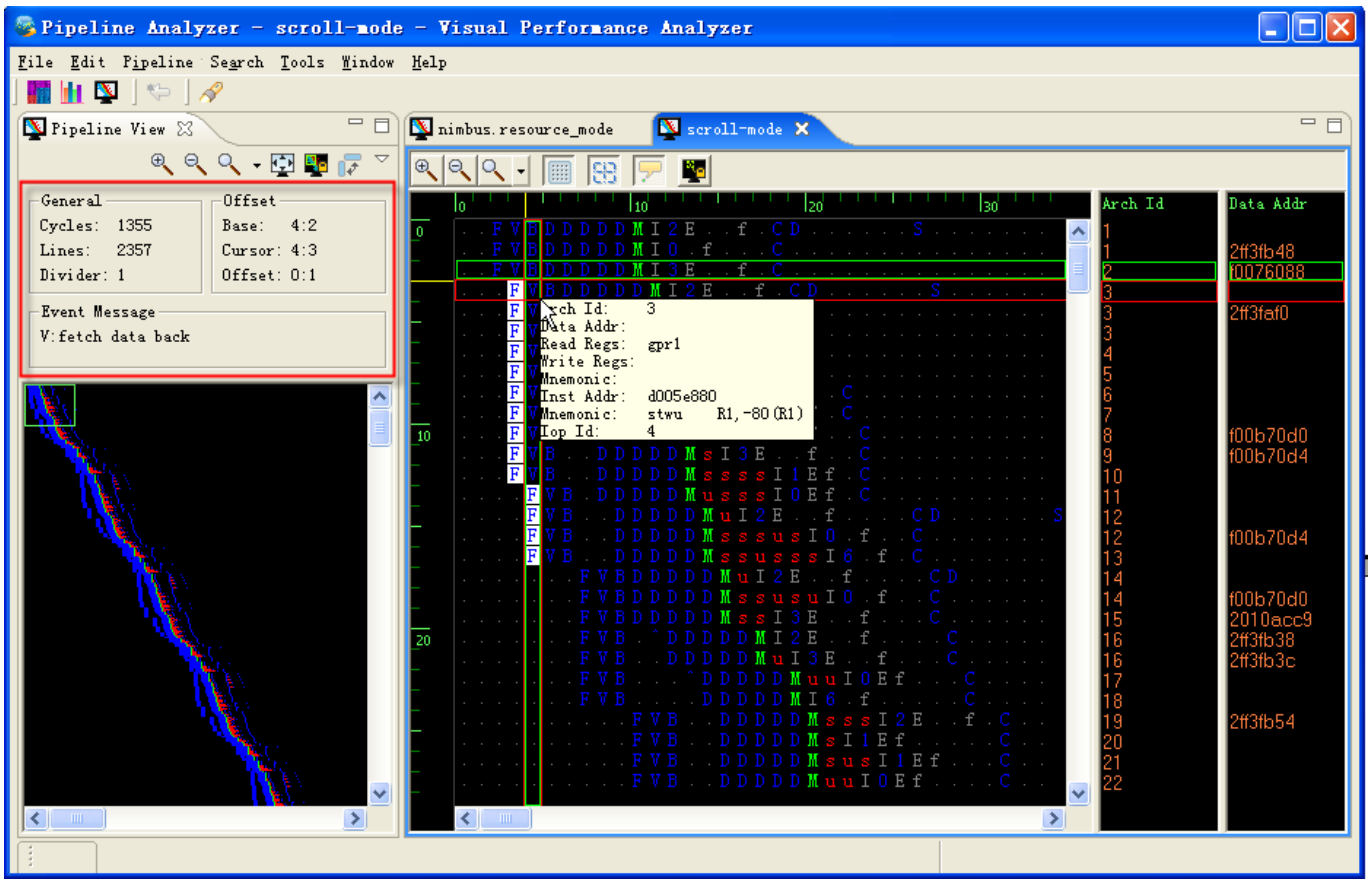


5.3.2.2Zoom in or out

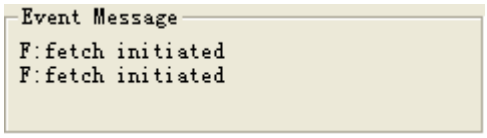
To zoom in or out this graph, press buttons in tool bar. To fit both width and height of the graph to overview panel, choose  in tool bar. Note that no scroll bars appear in overview panel after this operation.

5.3.2.3The event message and offset panel

The Event Message and Offset panel displays the denotation of instruction event which your mouse points at in the Scroll Editor. The following screen shot shows the change of this information when the mouse moves from symbol D to symbol M in the same line of table.

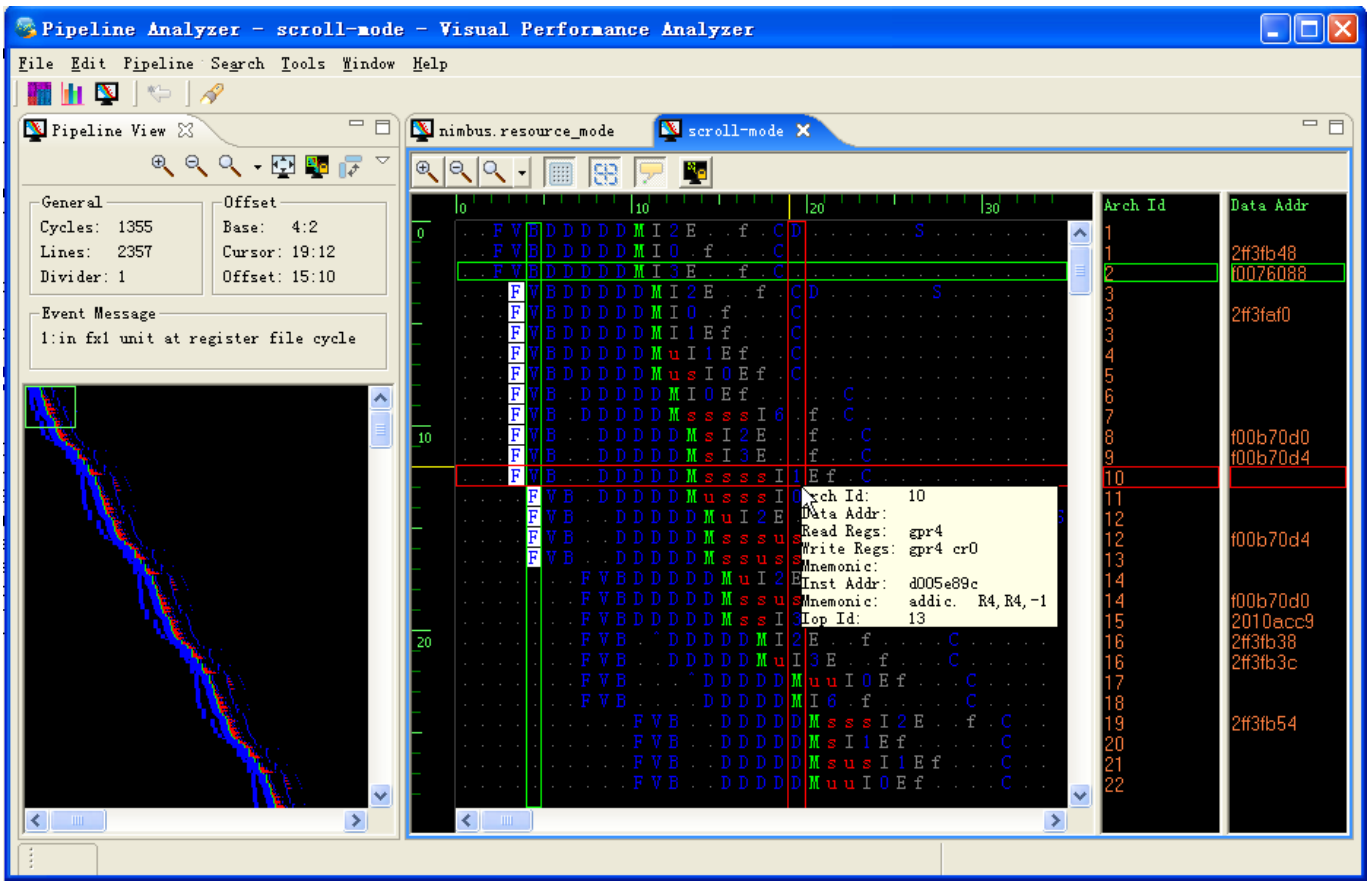


If two more events in an instruction execution occur at the same time cycle, their corresponding symbol will be highlighted and all the events be listed in Event Message panel. The following screen shot shows the event message of a highlighted symbol "F" in the above picture.



If there are too many event messages to display, you can resize this panel.

To scroll simultaneously with the currently active editor, press  in tool bar. The following screen shot shows this effect.

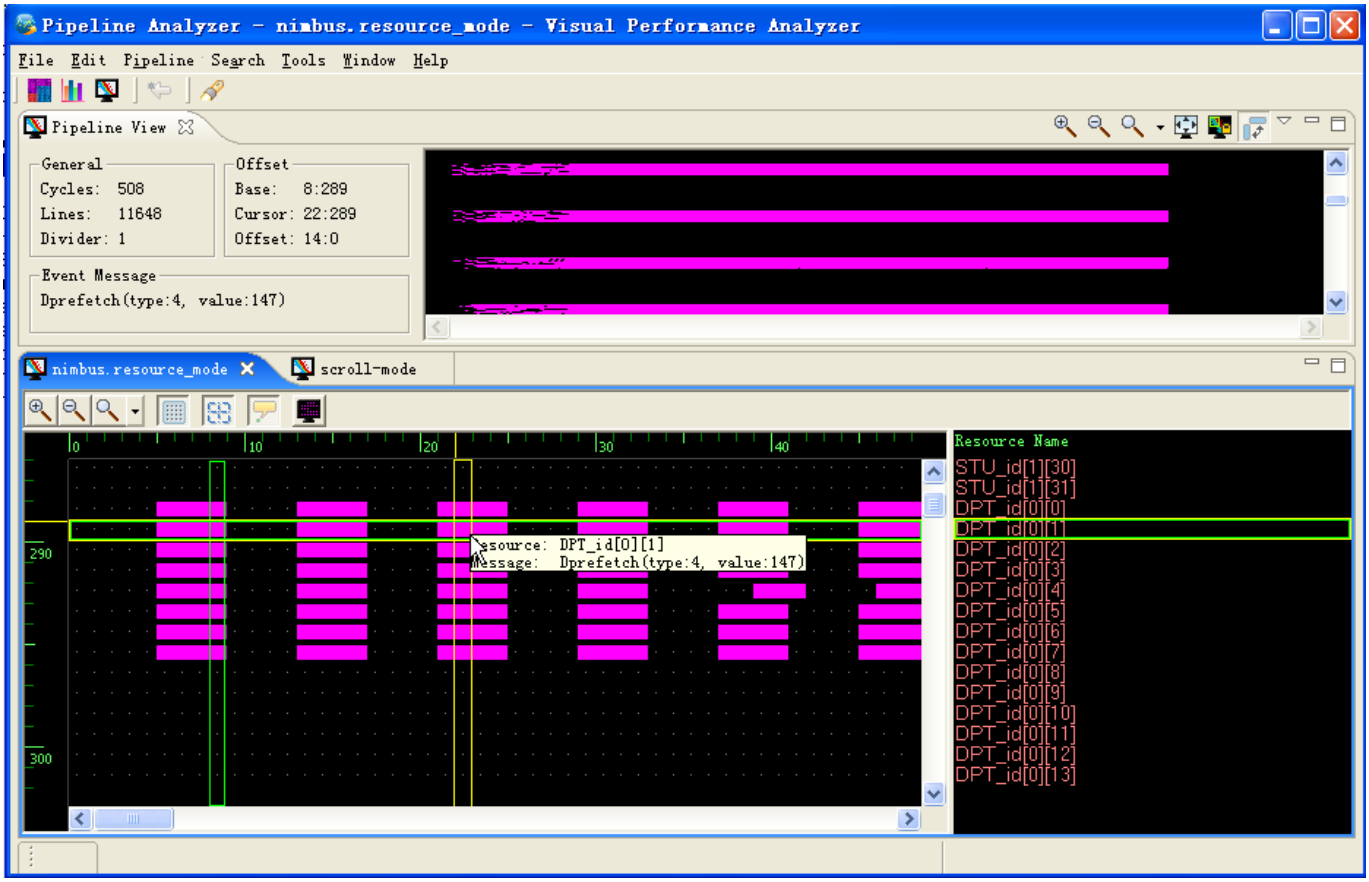


Please note that this function ensures that the green box is always within the overview graph of Pipe View.

5.3.3Navigating the resource view

To view pipeline file containing resource mode information, select **File -> Open File** and choose corresponding file. A resource editor opens in the Pipeline Analyzer Editor and Pipe View display its information at the same time.

The following screen shot shows the appearance of resource editor after a pipeline file is loaded.



In resource editor, a side bar named **Resource Name** in the right lists all the resources recorded in pipeline file. Each line of table in the left shows the usage distribution of this resource during time period. Each symbol in the table means an instruction event. Its denotation is shown in Event Message panel of Pipe View. If more than two events struggle for one resource at the same time, its symbol in the table turns red automatically. In this case, all the event messages are listed in Event Message panel of Pipe View. From General panel of Pipe View, you can see that this file has a total of 507 cycles and 11648 lines. The current time divider is 1, which means each symbol in the table indicates one time cycle.

The two red sliders in the table are named slider bar. It focuses on the cell which your mouse points at. Its ordinate is shown as Cursor value of Offset panel in Pipe View. The grey sliders in the table are named base axises. It focuses on the latest click of your left mouse. Its ordinate is shown as Base value of Offset panel in Pipe View. The distance between slider bar and base axises is calculated and shown as Offset value of Offset panel in Pipe View.

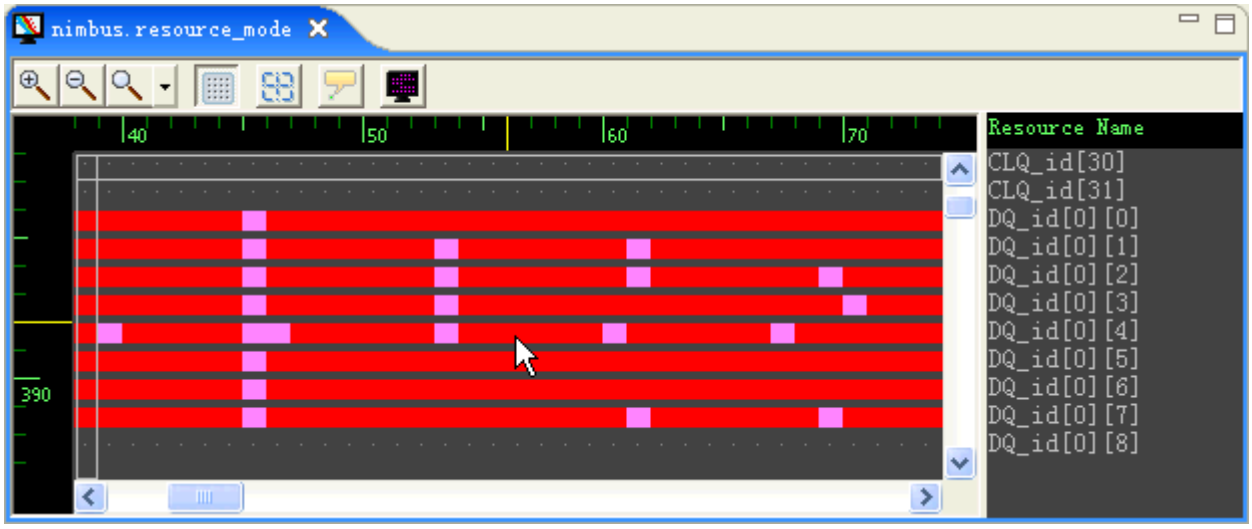
To navigate the editor, you can press left, ri0067ht, up, down keys or 'H', 'J', 'K', 'L' keys.

5.3.3.1Zoom in or out

To zoom in or out the table in scroll editor, press buttons in tool bar or select it in Pipeline Menu.

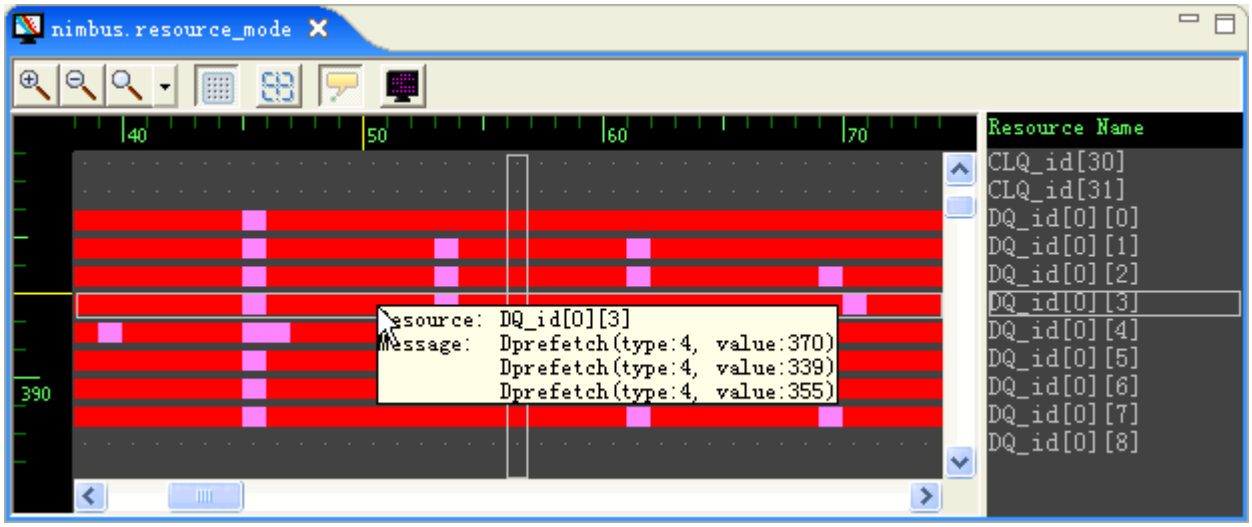
5.3.3.2Show Dots

To show dots in the table, press  in tool bar or select it in Pipeline Menu. The following picture shows a table with dots.



5.3.3.3Show Hover

To show the denotation of each symbol in the table, press in tool bar or select it in Pipeline Menu. While your mouse points at a line in the table for a while, a label which explains this line of resource usage turns up. The following screen shot shows the symbol message for a confile point in resource editor.



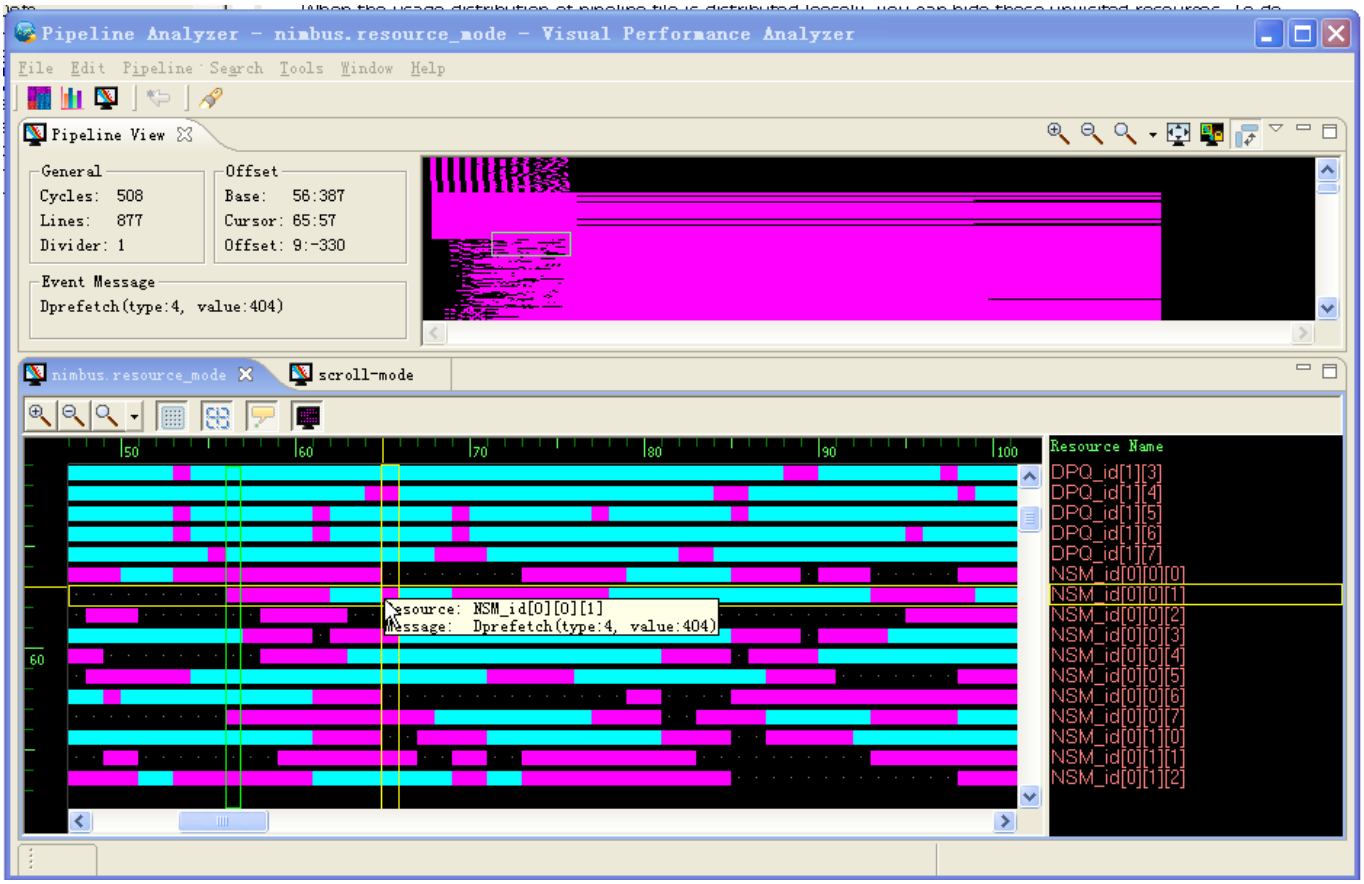
5.3.3.4Show Slider

To show slider bar while scrolling around the table, press in tool bar or select it in Pipeline Menu.

5.3.3.5Hide Unvisited Resources

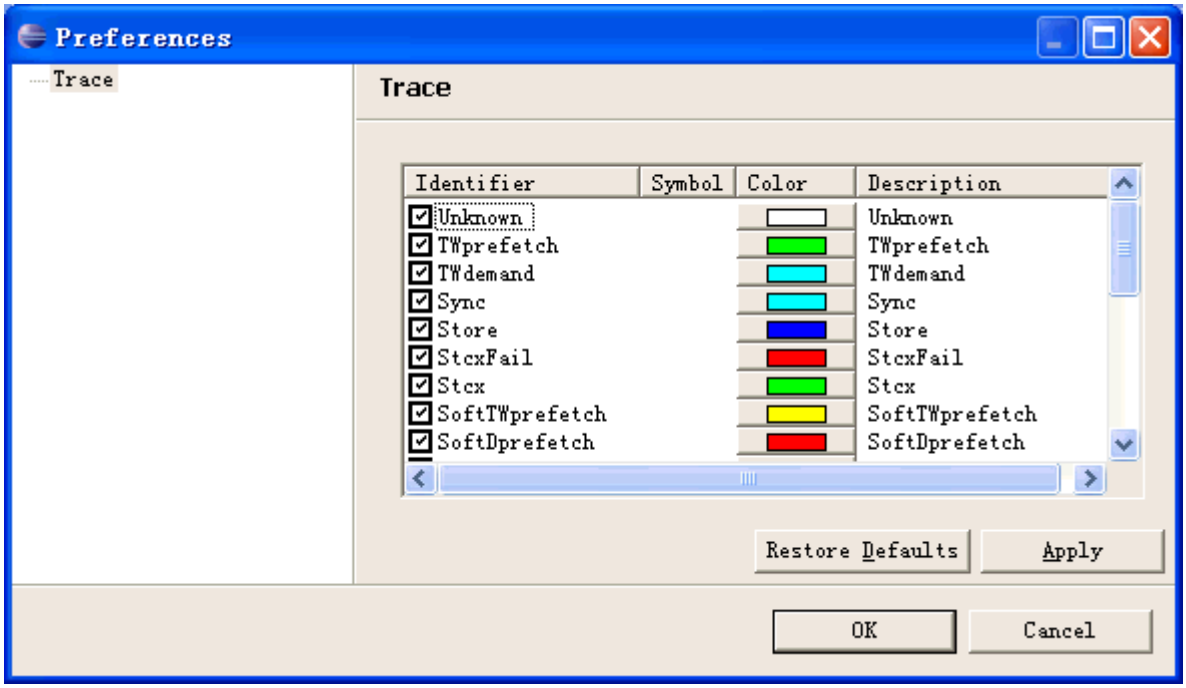
When the usage distribution of pipeline file is distributed loosely, you can hide those unvisited resources. To do this function, press in tool bar or select it in Pipeline Menu.

The following screen capture shows the condensed resource table for the above opened file.

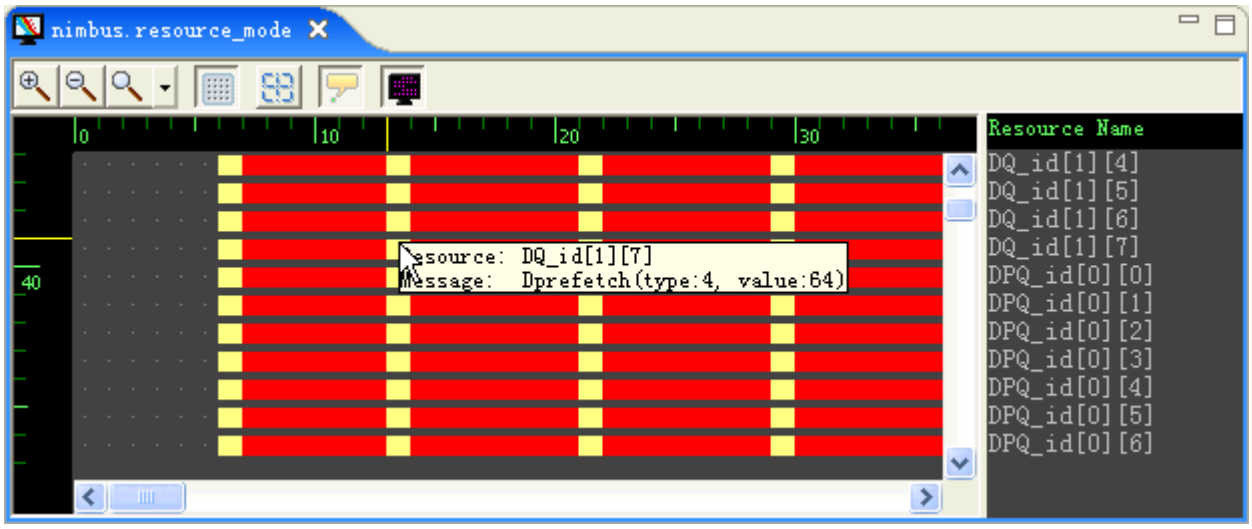


5.3.3.6Change Color for symbol

To change color for each symbol in the table, choose **Pipeline -> Settings... -> Trace.**



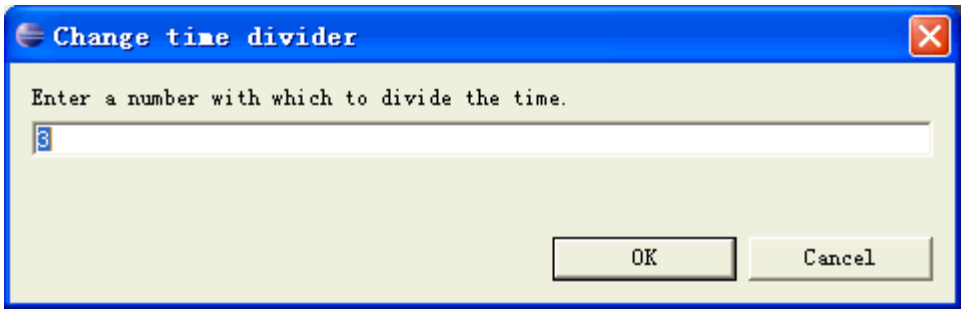
For example, if we set Dprefetch event in the opened resource file in the first screen shot to be yellow, the resource editor changes as follows:



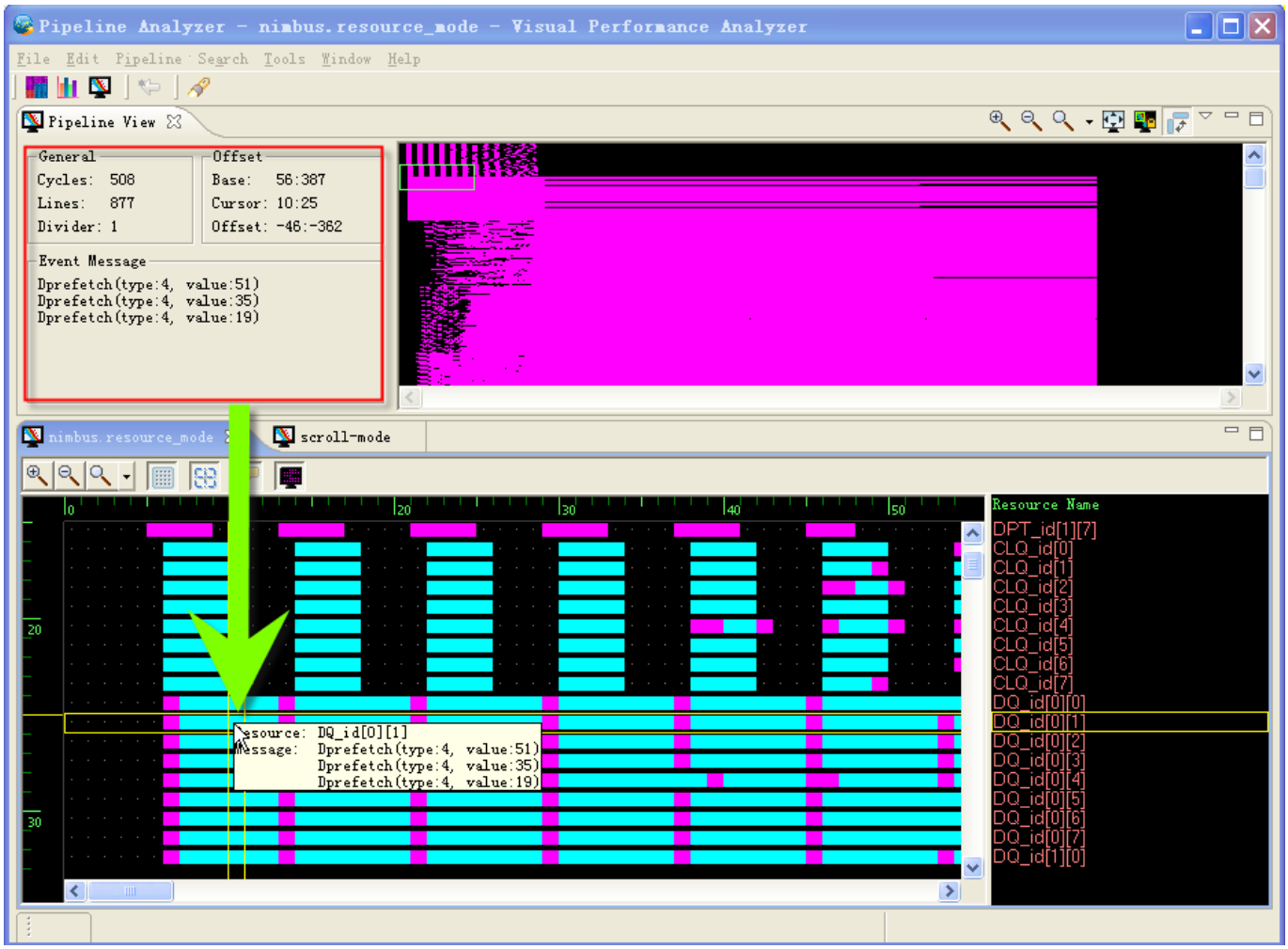
Please note, the red lines in this view means conflict for resource use. This is a system-defined color.

5.3.3.7Change Time Divider

To change the number of time cycles for each symbol in the table, choose **Pipeline -> Change Time Divider**.



The default value of this box is the current time divider. The following screen capture shows the resource editor after setting time divider to be three.



5.4Counter Analyzer

The Counter Analyzer tool is a common tool to analyze hardware performance counter data among many IBM eServer platforms, which includes systems running on AIX, i5OS, zOS, Linux on POWER, Linux on CELL/B.E..

The Counter Analyzer tool accepts hardware performance counter data in the form of a cross-platform XML file format. The tool uses either build-in hsqldb database engine or external DB2 instance to store the raw performance counter data. The tool provides multiple views to help user identify the data. The views can be divided into two categories: one category is the “table” views, which are basically two-dimension tables displaying data. The data could be raw performance counter values, derived metrics, counter comparison results and so on. Another category is the “plot” views. In these views data are represented by different kind of plots. The data could also be raw performance counter values, derived metrics, and comparison results and so on. Besides these “table” views and “plot” views, there are also some “utility” views to help user configure and customize the tool.

You can also find the Pipeline Analyzer User Guides from the VPA. Select **Help - Help Contents** within VPA. To get context sensitive help, press **F1** for Windows and AIX or press **Ctrl+F1** for Linux.

5.4.1Basic concepts for Counter Analyzer

- **Performance Monitoring Counter**

Performance monitor counter provides comprehensive reports of events that are critical to performance on IBM systems. It is able to gather critical hardware events, such as the number of misses on all cache levels, the number of floating point instructions executed, the number of instruction loads that cause TLB misses.

- **Metrics**

Metric is calculated with user-defined formula and event count from performance monitor counter. It's used to provide performance information like CPU utilization rate, million instructions per second. This helps the algorithm designer or programmer identify and eliminate performance bottlenecks.

- **CPI Breakdown Model**

Cycles per instruction(CPI) is the measurement for analyzing the performance of a workload. CPI is simply defined as the number of processor clocked cycles needed to complete an instruction. It is calculated as CPI = Total Cycles / Number of Instructions Completed. A high CPI value usually implies underutilization of machine resources.

On a POWER5 system, you can break down your workload CPI into individual components, as the POWER5 has several programmable counters available to count events that can calculate the components of CPI and allow you to determine how to improve performance on a given workload.

The following is an instance of CPI breakdown model:

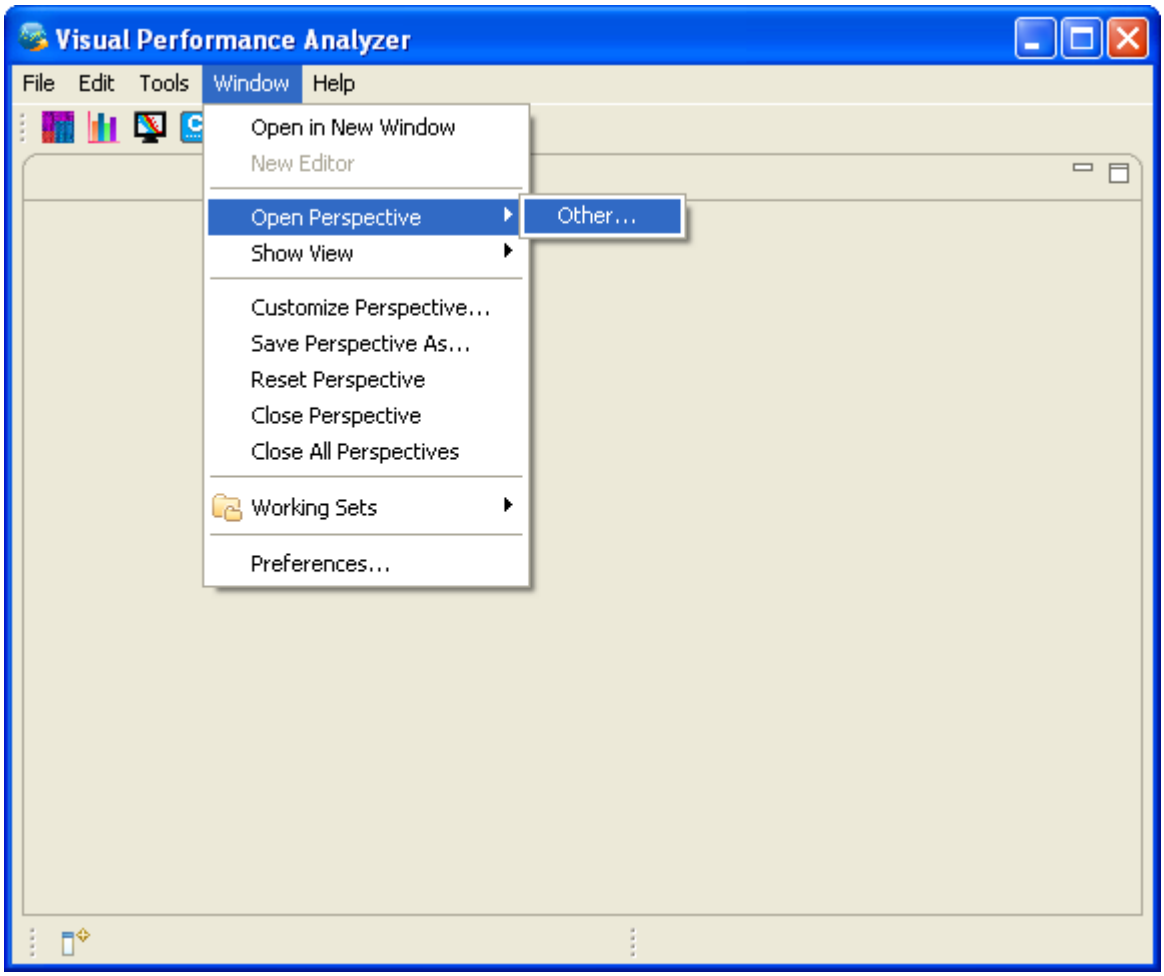
Total cycle <# cycles>	Completion cycles <A:group complete cycles>	PowerPC Base completion cycles <A1: One or more PowerPC instructions completed this cycle>		
		overhead of cracking/microcoding and grouping restriction <A2:(A)-(A1)>		
	Completion Table empty (GCT empty) cycles <B>	I-cache miss penalty <B1>		
		Branch redirection (branch misprediction) penalty <B2>		
		others (Flush penalty etc.) <B4: (B)-(B1)-(B2>		
	Completion Stall cycles <C: total-(A)-(B)>	Stall by LSU inst <C1>	Stall by reject <C1A>	Stall by translation (rejected by ERAT miss) <C1A1>
				Other reject <C1A2: (C1A)-(C1A1)>
			Stall by D-cache miss <C1B>	
			Stall by LSU basic latency, LSU Flush penalty <C1C: (C1)-(C1A)-(C1B)>	
		Stall by FXU inst <C2>	Stall by any form of DIV/MTSPR/MFSPR inst <C2A>	
			Stall by FXU basic latency <C2C: (C2)-(C2A)>	
		Stall by FPU inst <C3>	Stall by any form of FDIV/FSQRT inst <C3A>	
			Stall by FPU basic latency <C3B: (C3)-(C3A)>	
		others (Stall by BRU/CRU inst , flush penalty (except LSU flush), etc.) <C4: (Completion Stall cycles)-(C1)-(C2)-(C3) >		

The above table represents a CPI breakdown model where the total cycles of a workload is divided into three components: Completion cycles, Completion Table empty or GCT empty cycles, and Completion Stall cycles. The base completion cycles are the number of cycles that would be needed if grouping was perfect. Otherwise, stalls happen, and they can be attributed to either Completion Table empty or Completion Stall cycles. A Completion Table empty condition occurs when no groups are completing on a given cycle because of either Icache miss or branch misdirection. Meanwhile, the Completion Stall cycles are those stalls caused by any of the following instructions: LSU, FXU, FXU long (all forms of div, mtspr, mfspr), FPU, and FPU long (all forms of fsqrt, fdiv); or by other events such as Dcache miss, Reject, and Reject by translating (ERAT miss).

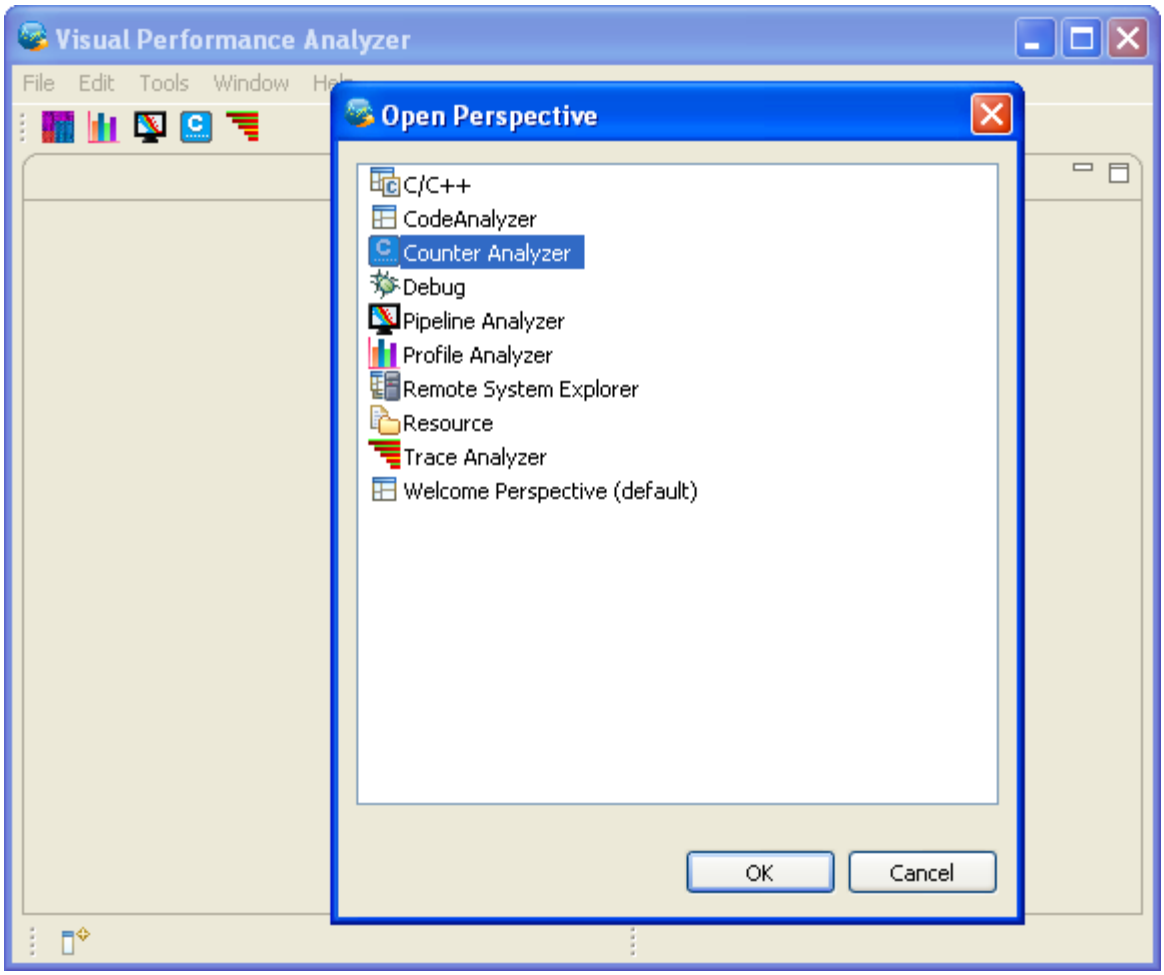
5.4.2Load an existing counter data file

5.4.2.1Open Counter Analyzer Perspective

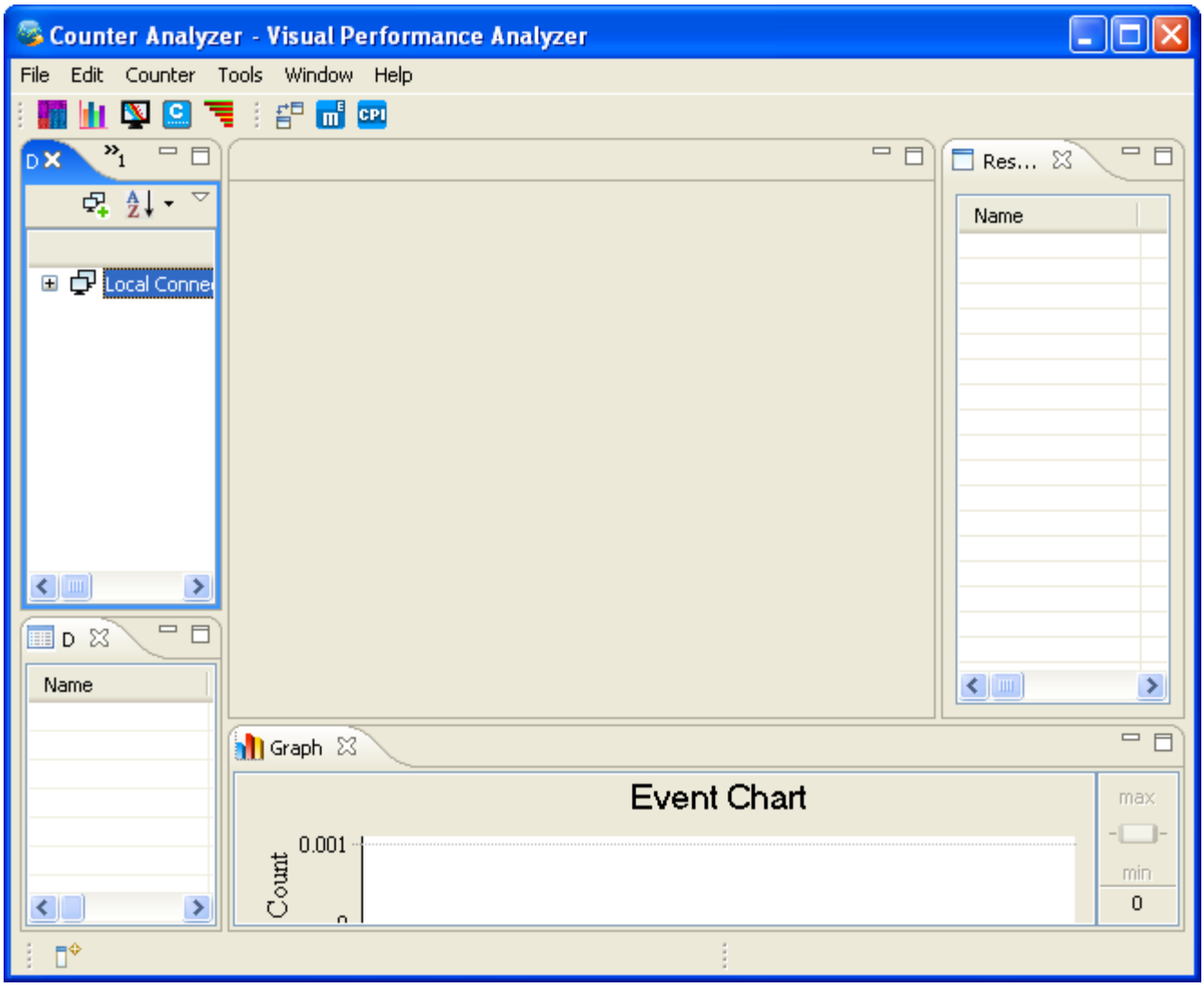
1. When you first start Visual Performance Analyzer after installation, the default perspective is Profile Analyzer. To start Counter Analyzer, choose **Window -> Open Perspective -> Other.**



2. In the Select Perspective dialog choose **Counter Analyzer** and click **OK**.



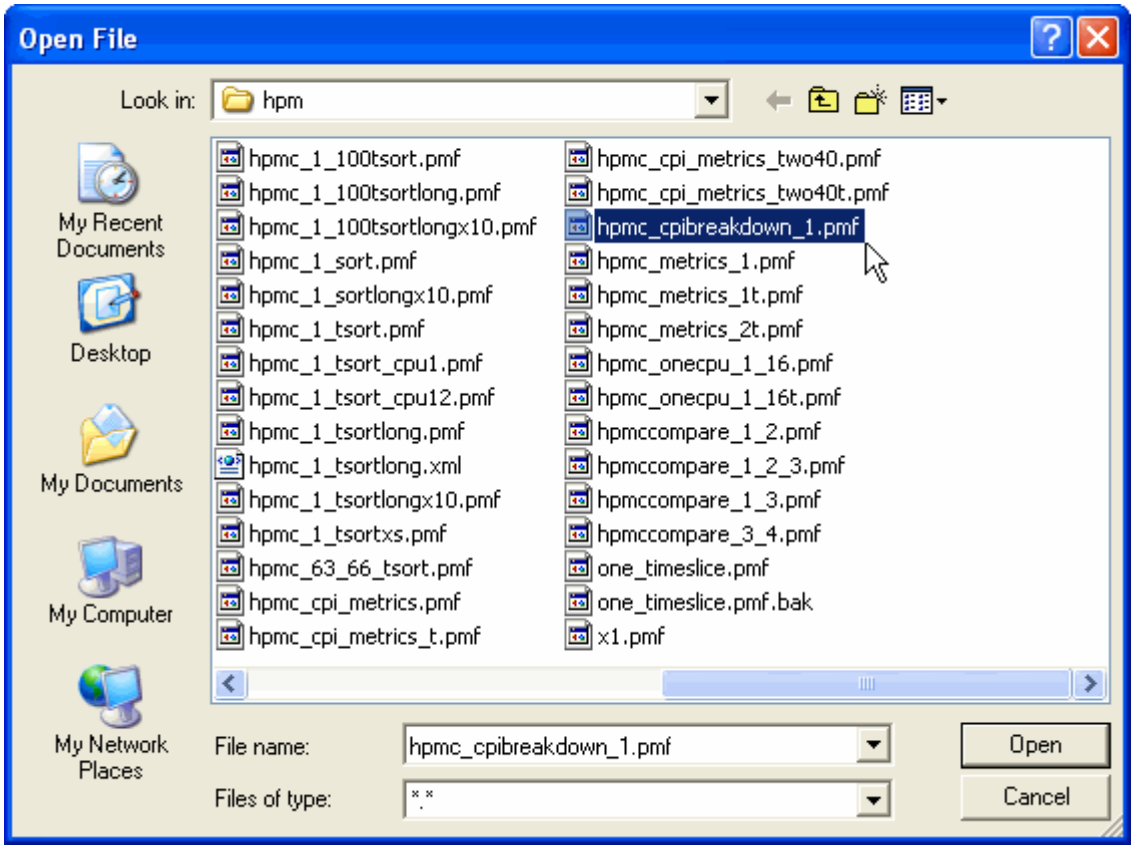
3. The following screen capture shows the initial layout of Counter Analyzer.



4. You can also open **Counter Analyzer** Perspective simply by clicking in the toolbar.

5.4.2.2Load in counter data file

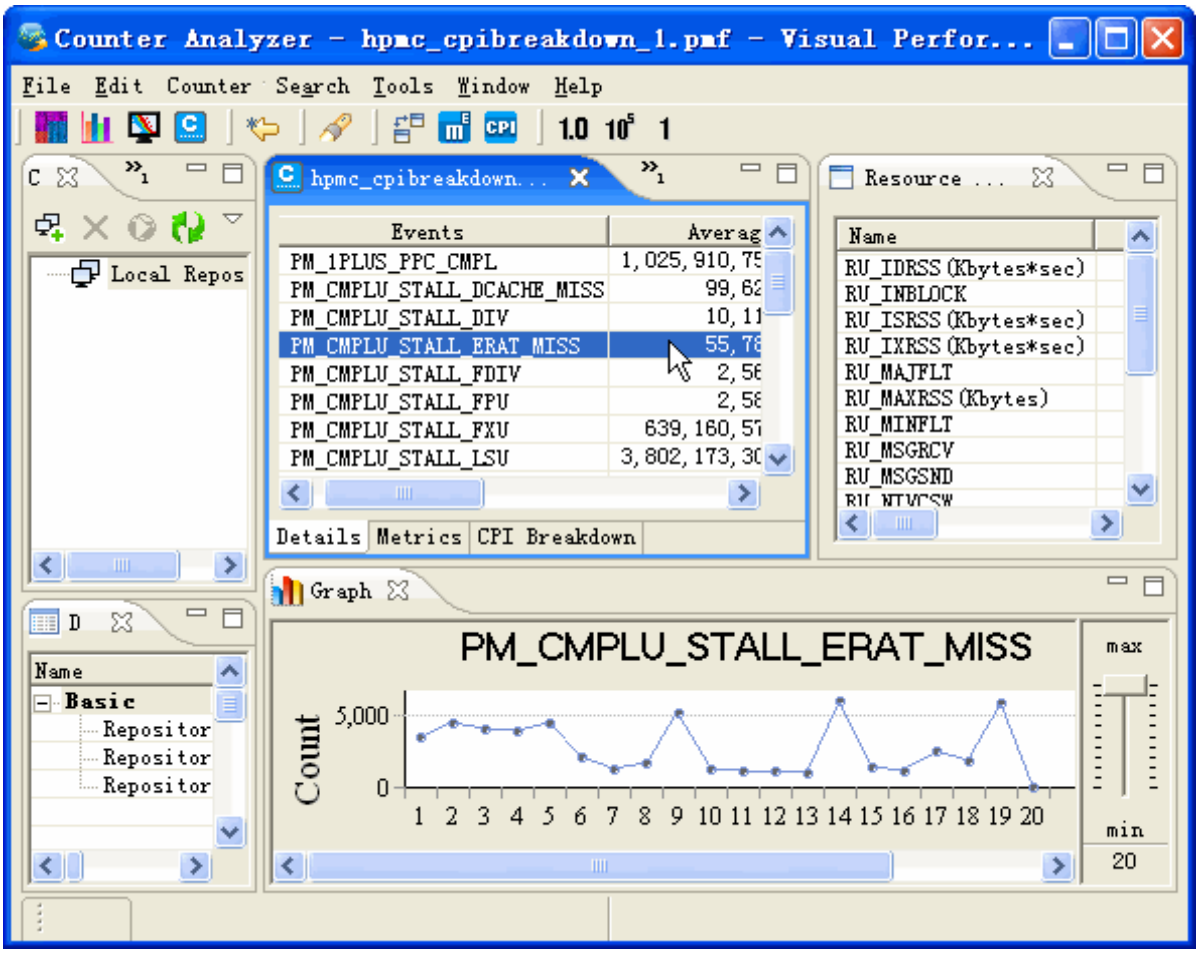
Choose **File -> Open File**_, and in Open File dialog select one counter data file with suffix ".pmf".



You can also load in counter data from repositories, which will not be covered here.

5.4.2.3 Brief introduction to Counter Analyzer Perspective

After loading in the counter data of .pmf file, the Counter Analyzer Perspective displays the data in its views and editors. Primary information of details, metrics and CPI breakdown is displyed in Counter Editor. Resource statistics information of the file (if available) will be showed in tabular view **Resource Statistics**. View **Graph** illustrates the details, metrics and CPI breakdown in a graphic way.



5.4.3Navigate the Counter Analyzer Perspective

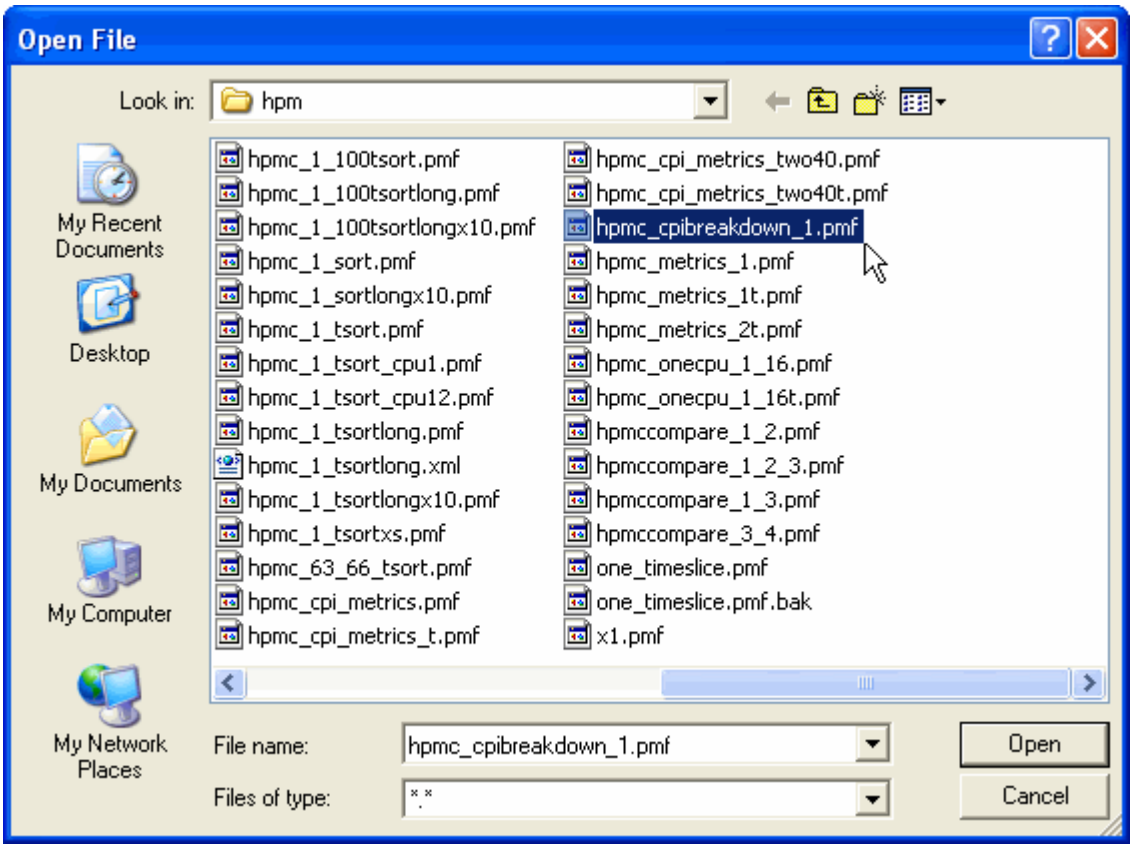
The following are tasks that you can perform to navigate around counter data within Counter Analyzer.

5.4.3.1Open Counter Data

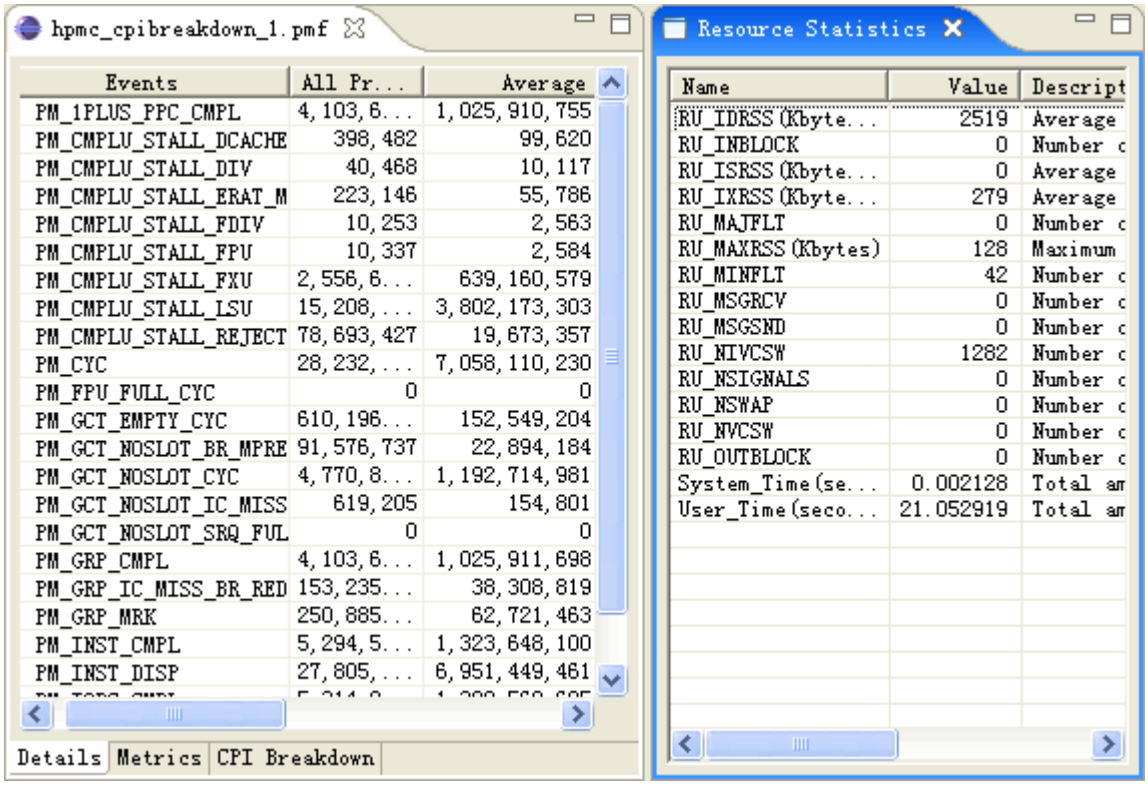
There are two ways to open counter data using Counter Analyzer, either from counter data files or from database repository.

5.4.3.1.1**Open Counter Data from PMF Files**

1. Choose **File -> Open File**_, and in Open File dialog select one counter data file with suffix ".pmf".



2. If one counter file is selected, it will be opened in editor, and its resource statistics information (if available) will be showed in tabular view **Resource Statistics**.



By default, a local repository is provided by VPA tool. You can also connect to remote DB2 repositories. Remote repositories can be created, configured, refreshed, and discarded. Files in these repositories can be opened in

Counter Editor and deleted. Besides, you can import counter data files into and query counter data from these repositories.

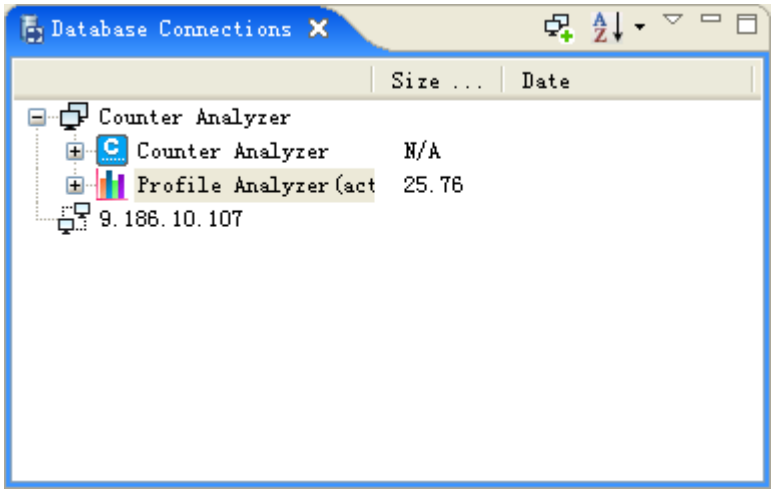
5.4.3.1.2Open Counter Data From Connection View

By default, a local repository is provided by VPA tool. You can also connect to remote DB2 repositories. The local connection is a local connection, which you can add Counter Analyzer Support under it, as well as other supports. Remote connection is a DB2 connection. It can be created, configured, refreshed, and deleted.

- **Create/Configure/Refresh/Discard a Connection**

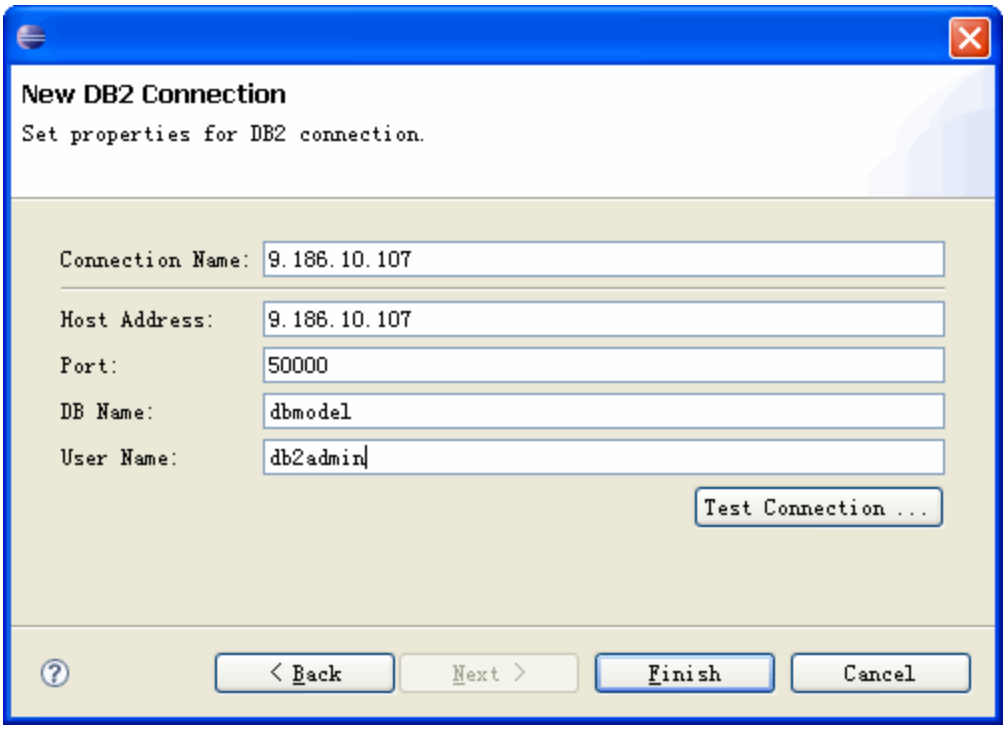
Create Connection

In **Database Connection** view, select **New Connection** in the context menu, you can create a new repository. Local Repository is created by default and can only be configured and refreshed. **Description** view lists basic information of the repository.



Configure Connection

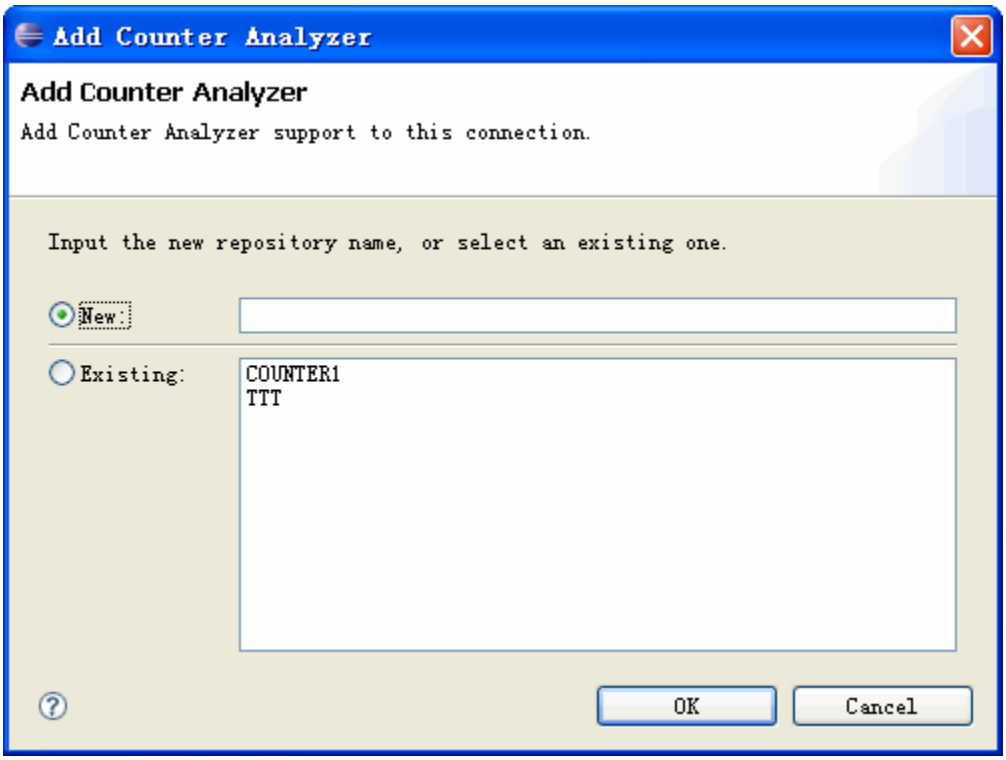
The following is the Configure Dialog of Connection.



Add Product Support

Right-click on the connection node, choose the product support you want to add to the connection node, a confirm dialog will occur.

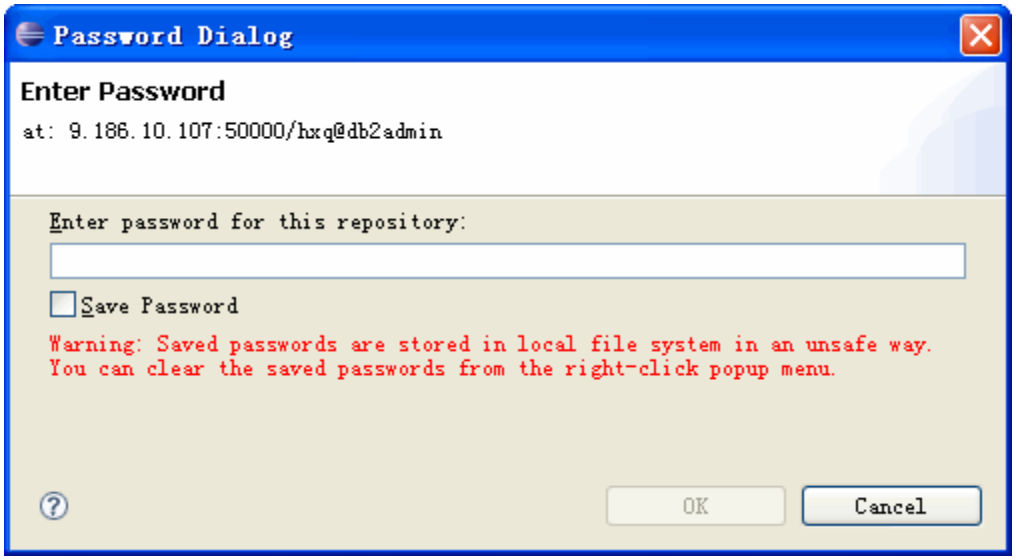
You can create a new repository, or select an existing repository from the Existing List.



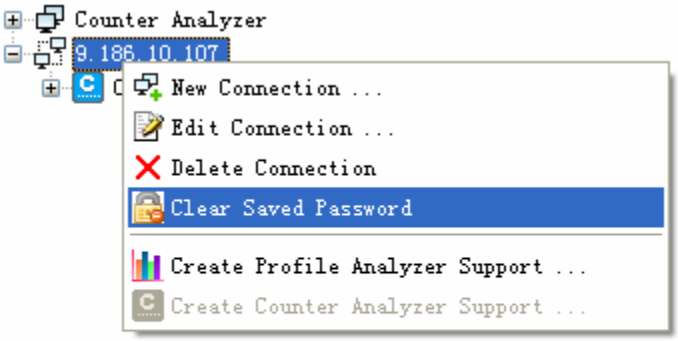
Refresh Connection

Double click a Connection or click on the + icon to expand the tree.

A **Password Dialog** will pop up when opening the repository for the first time. If the password is saved, you no longer have to re-enter it when Counter Analyzer is closed and restarted later. If the password is not saved, you are required for it every time they start Counter Analyzer.

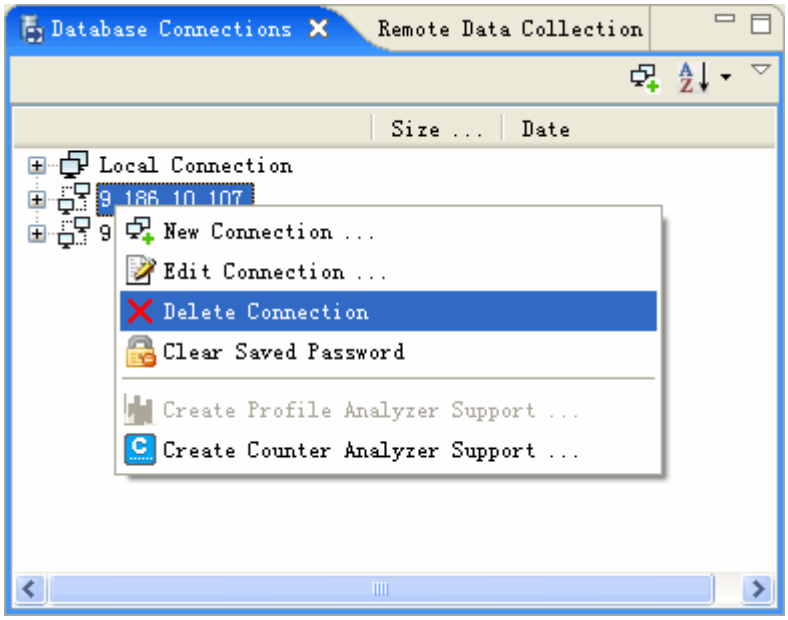


You can also choose to clear the password on the context menu.



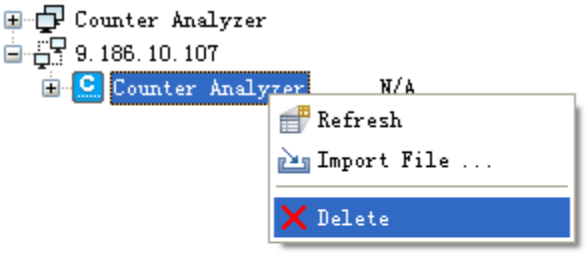
Discard Connection

You can choose to discard the Connection on the context menu. Be sure that delete all its children first.



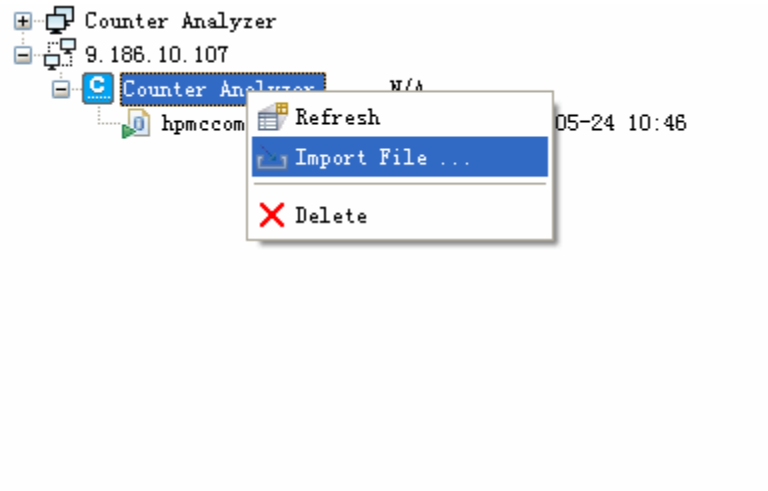
Discard Support

You can choose to discard the Product Support on the context menu.



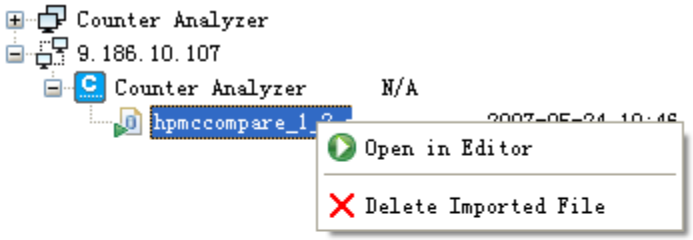
- **Import File**

Right-click on the Support node, select Import File action, then, choose the file you want to import from the file dialog.



➢ **Open File in Editor**

Right-click on the File node, select Open in Editor, or double-click on the file.

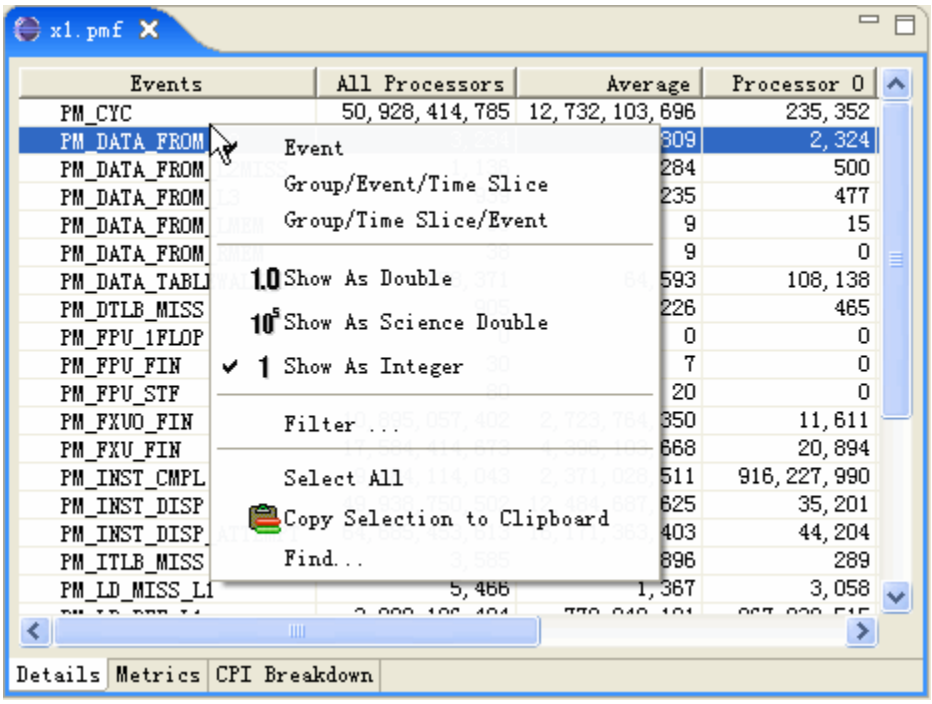


5.4.3.2View Counter Data

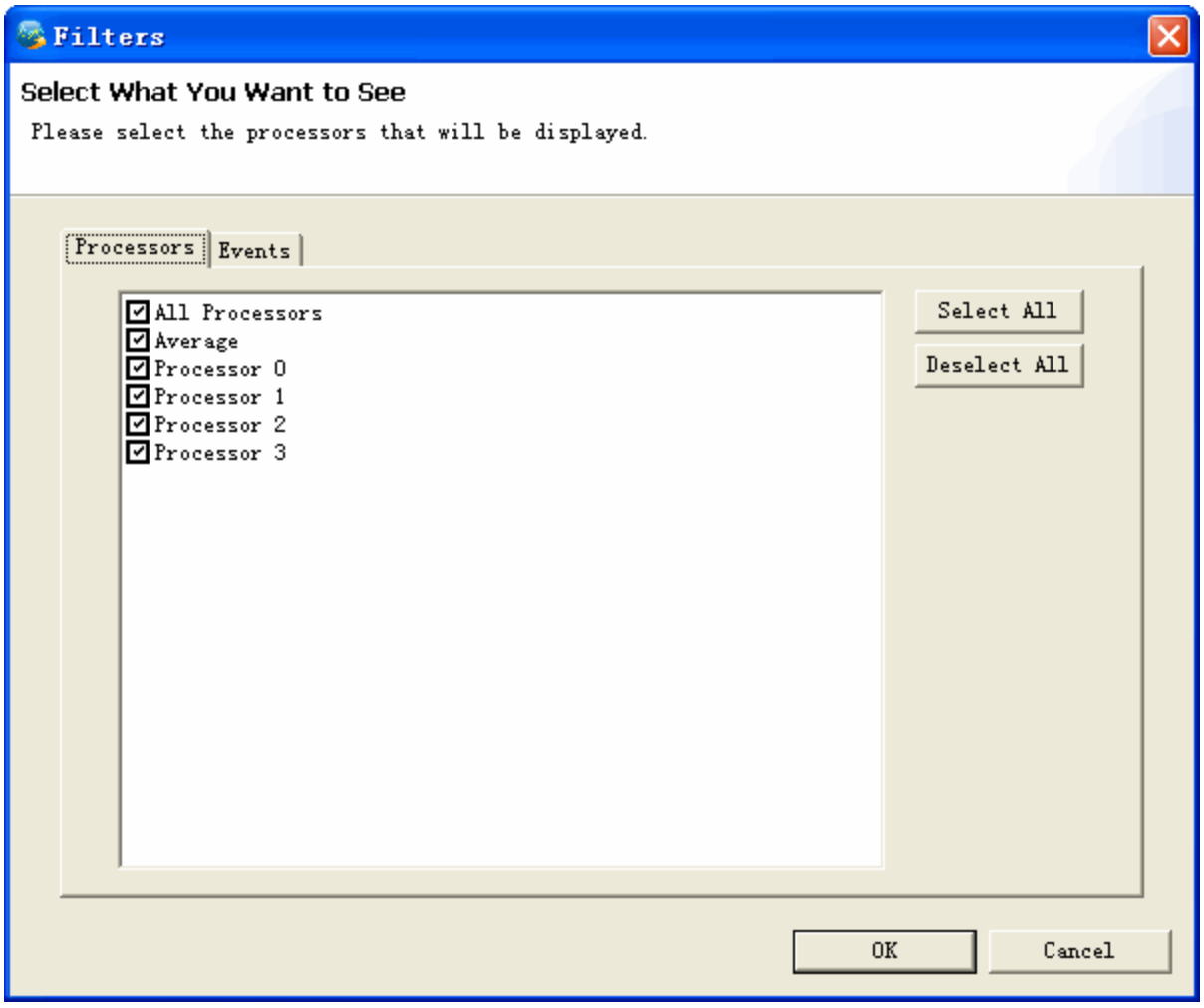
You can see raw counter data, metrics data and CPI breakdown data in different pages of editor. All these data are retrieved from one counter file.

5.4.3.2.1View Raw Counter Data

Row counter data are shown In Tab **Details**. On the context menu, you can switch display mode here.



In all tabs, **Filter** is used to filter processors and events.



Row counter data can be displayed in three modes: **Event**, **Group/Event/Time Slice**, and **Group/Time Slice/Event**. If the counter data has time slice information, all these three modes can be supported. If not, only **Event** mode is enabled, and the other two modes are disabled.

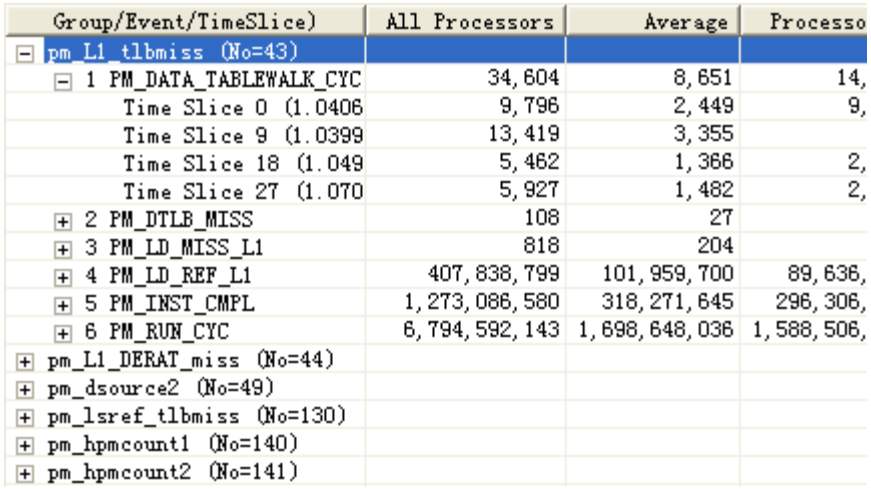
"Event" Mode

In this mode, all events and their counter data are listed in the editor. Data here are normalized event count instead of actual data in counter data file.

Events	All Processors	Average	Processor 0
PM_CYC	50,928,414,785	12,732,103,696	235,352
PM_DATA_FROM_L2	3,234	809	2,324
PM_DATA_FROM_L2MISS	1,136	284	500
PM_DATA_FROM_L3	939	235	477
PM_DATA_FROM_LMEM	38	9	15
PM_DATA_FROM_RMEM	38	9	0
PM_DATA_TABLEWALK_CYC	258,371	64,593	108,138
PM_DTLB_MISS	905	226	465
PM_FPU_1FLOP	0	0	0
PM_FPU_FIN	30	7	0
PM_FPU_STF	80	20	0
PM_FXUO_FIN	10,895,057,402	2,723,764,350	11,611
PM_FXU_FIN	17,584,414,673	4,396,103,668	20,894
PM_INST_CMPL	9,484,114,043	2,371,028,511	916,227,990
PM_INST_DISP	49,938,750,502	12,484,687,625	35,201
PM_INST_DISP_ATTEMPT	64,685,453,613	16,171,363,403	44,204

"Group/Event/Time Slice" Mode

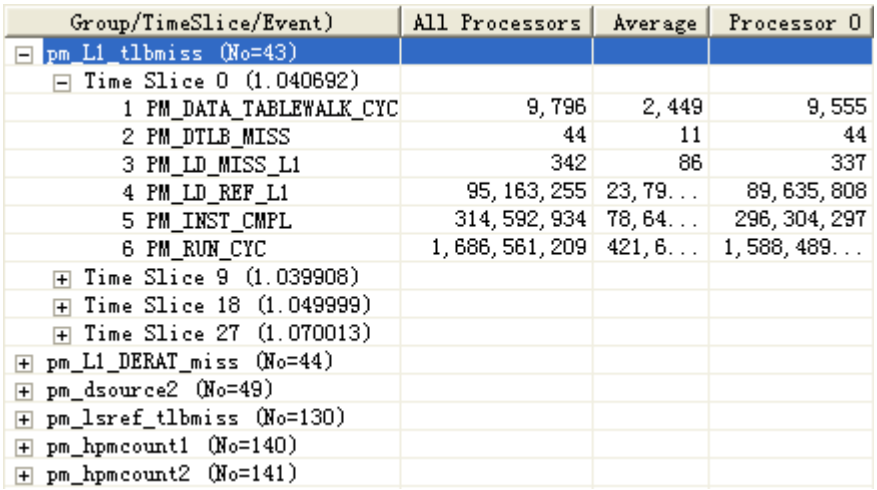
If the counter data has time slice information, there are two modes to display more. In **Group/Event/Time Slice** mode, the data can be grouped first by group, and then event. Data here are actual event count in counter data file.



Group/Event/TimeSlice)	All Processors	Average	Processo
⊟ pm_L1_tlbmiss (No=43)			
⊟ 1 PM_DATA_TABLEWALK_CYC	34,604	8,651	14,
Time Slice 0 (1.0406	9,796	2,449	9,
Time Slice 9 (1.0399	13,419	3,355	
Time Slice 18 (1.049	5,462	1,366	2,
Time Slice 27 (1.070	5,927	1,482	2,
⊞ 2 PM_DTLB_MISS	108	27	
⊞ 3 PM_LD_MISS_L1	818	204	
⊞ 4 PM_LD_REF_L1	407,838,799	101,959,700	89,636,
⊞ 5 PM_INST_CMPL	1,273,086,580	318,271,645	296,306,
⊞ 6 PM_RUN_CYC	6,794,592,143	1,698,648,036	1,588,506,
⊞ pm_L1_DERAT_miss (No=44)			
⊞ pm_dsource2 (No=49)			
⊞ pm_lsref_tlbmiss (No=130)			
⊞ pm_hpmcount1 (No=140)			
⊞ pm_hpmcount2 (No=141)			

"Group/Time Slice/Event" Mode

If the counter data has time slice information, the other mode to display more information is **Group/Time Slice/Event mode**, in which counter data is grouped first by group, and then time slice. Data here are also actual event count in counter data file.



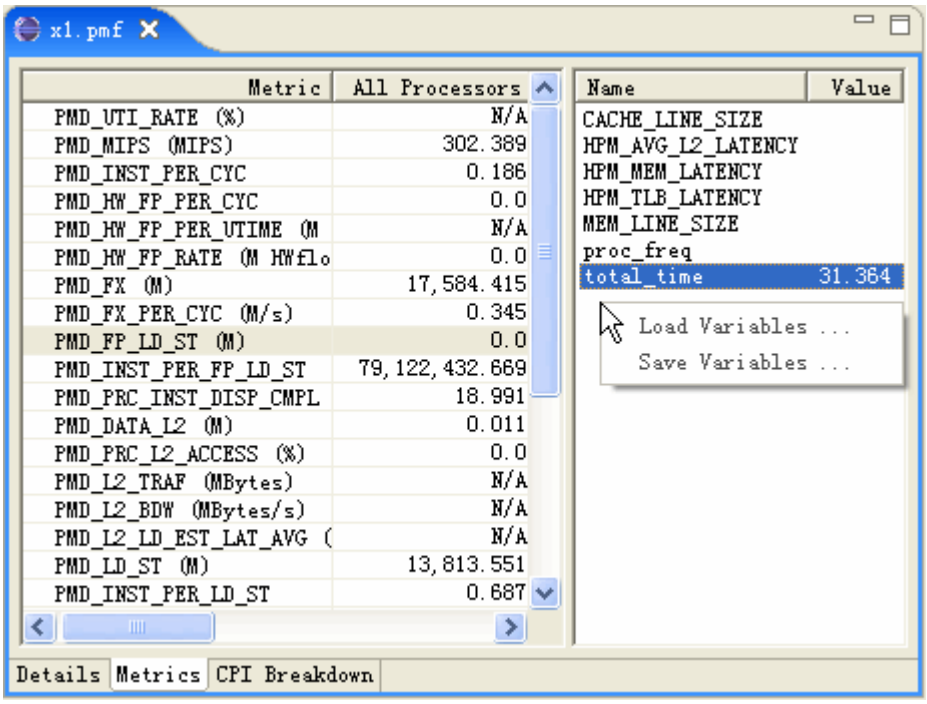
Group/TimeSlice/Event)	All Processors	Average	Processor 0
⊟ pm_L1_tlbmiss (No=43)			
⊟ Time Slice 0 (1.040692)			
1 PM_DATA_TABLEWALK_CYC	9,796	2,449	9,555
2 PM_DTLB_MISS	44	11	44
3 PM_LD_MISS_L1	342	86	337
4 PM_LD_REF_L1	95,163,255	23,79...	89,635,808
5 PM_INST_CMPL	314,592,934	78,64...	296,304,297
6 PM_RUN_CYC	1,686,561,209	421,6...	1,588,489...
⊞ Time Slice 9 (1.039908)			
⊞ Time Slice 18 (1.049999)			
⊞ Time Slice 27 (1.070013)			
⊞ pm_L1_DERAT_miss (No=44)			
⊞ pm_dsource2 (No=49)			
⊞ pm_lsref_tlbmiss (No=130)			
⊞ pm_hpmcount1 (No=140)			
⊞ pm_hpmcount2 (No=141)			

5.4.3.2.2View Metrics Data

Metrics data are shown in Tab **Metrics**.

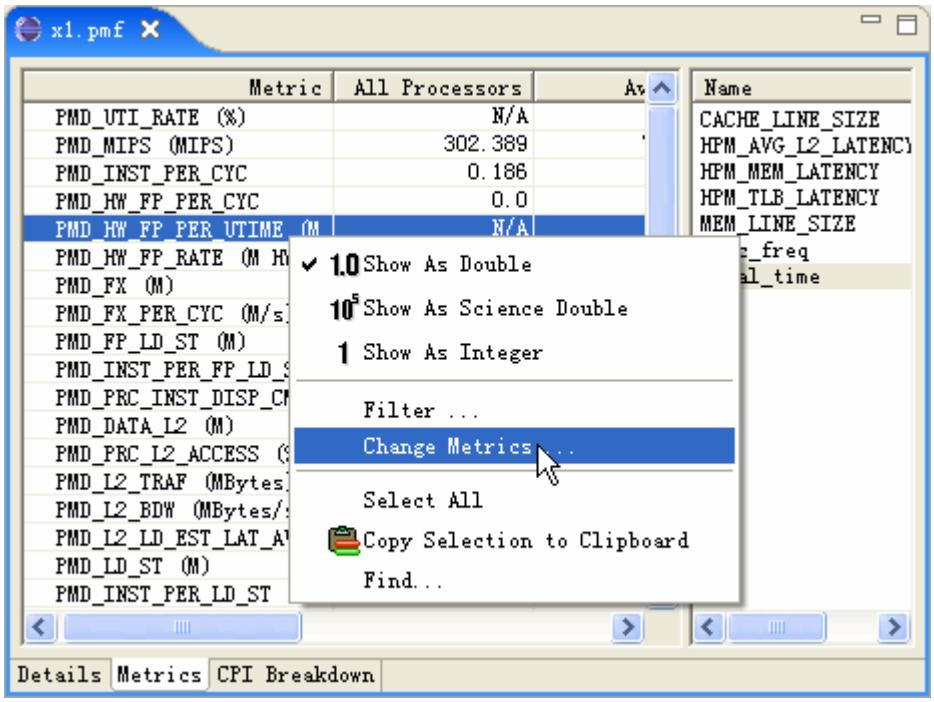
Edit/Load/Save Metrics Variables

In the right part of the editor, all variables associated with the current metrics are listed in a table with their names and values. Left click the **Value** column of each variable, and you can modify its value. On the context menu, users can further choose **Load Variables** to load one variables file, and choose **Save Variables** to save current variables in the editor to one variables file. If one variables file is loaded, all variables' values are updated, which makes all metrics be calculated again. (Only Variable "total_time" is read only and cannot be overwritten.)

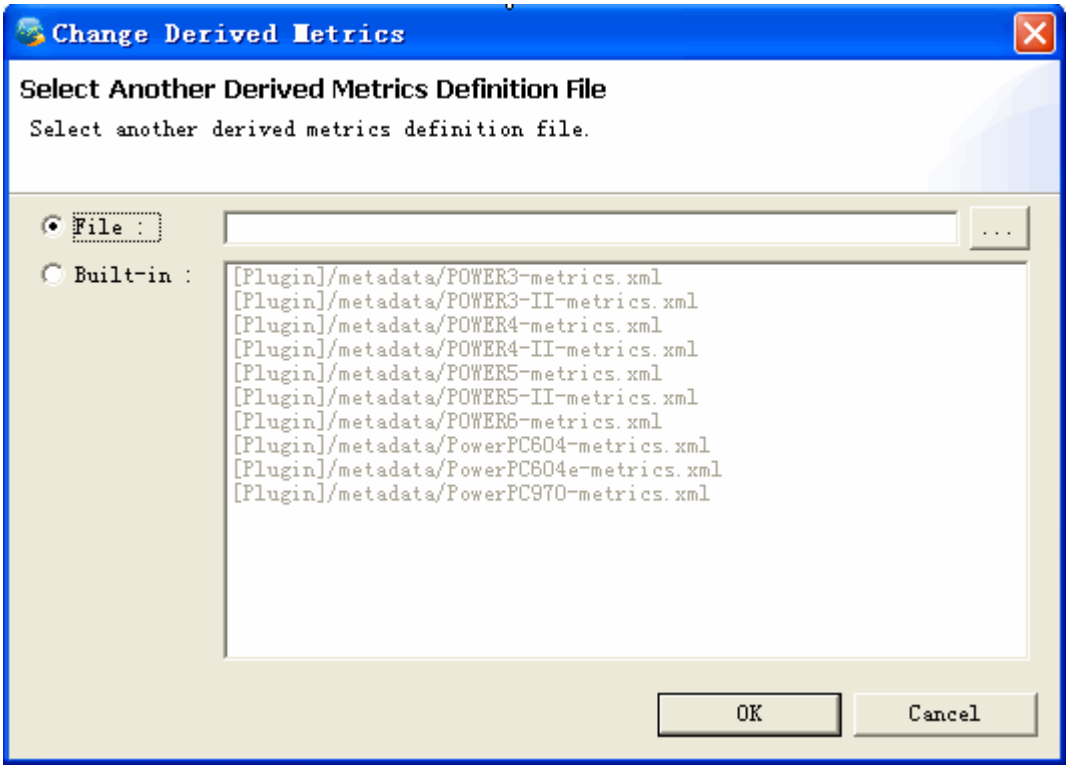


Change Metrics

You can choose to change metrics file on the context menu, and applies it to the active counter data.



The metrics file to be selected can be either external files or built-in files. The change derived metrics dialog is as follows:

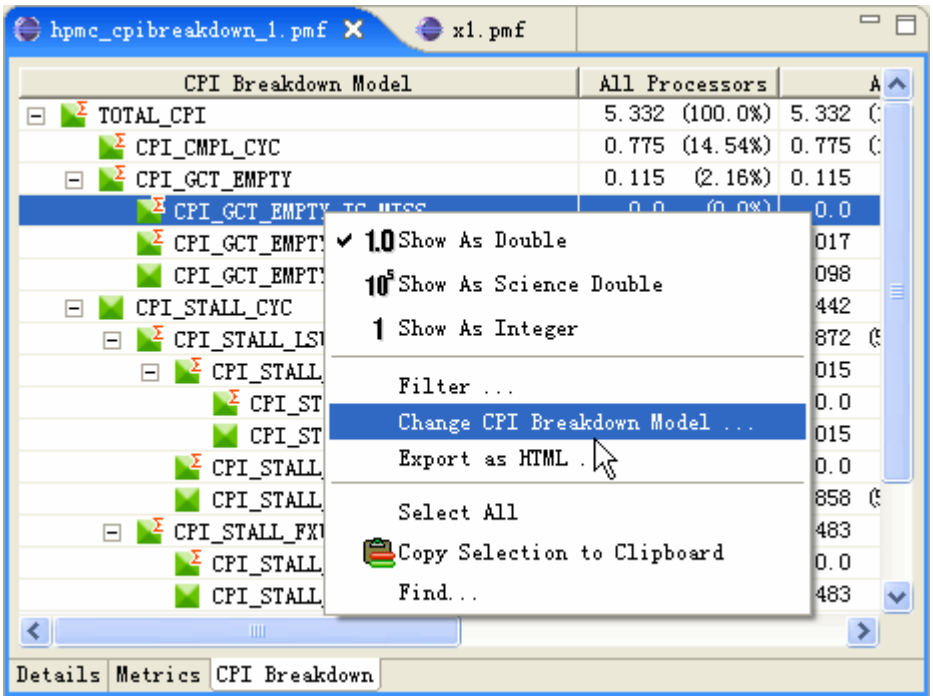


5.4.3.2.3View CPI Breakdown Data

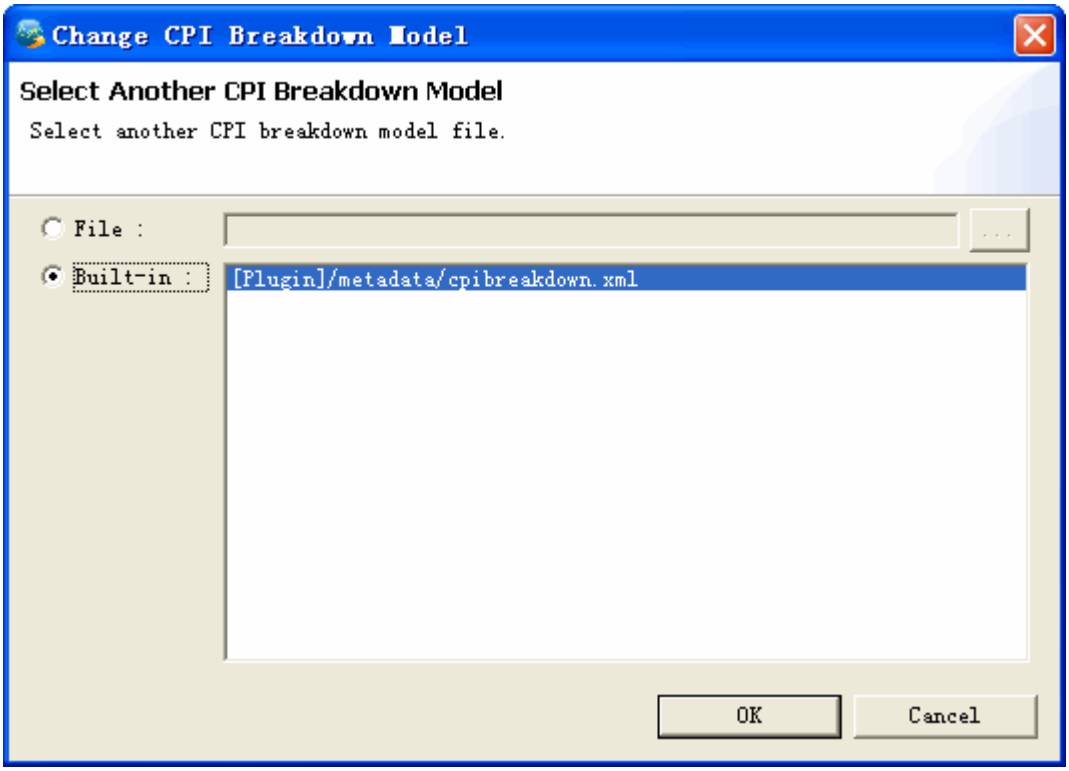
CPI breakdown data are shown in Tab **CPI Breakdown**.

Change CPI Breakdown Model

You can change CPI breakdown model on the context menu, and apply it to the active counter data.

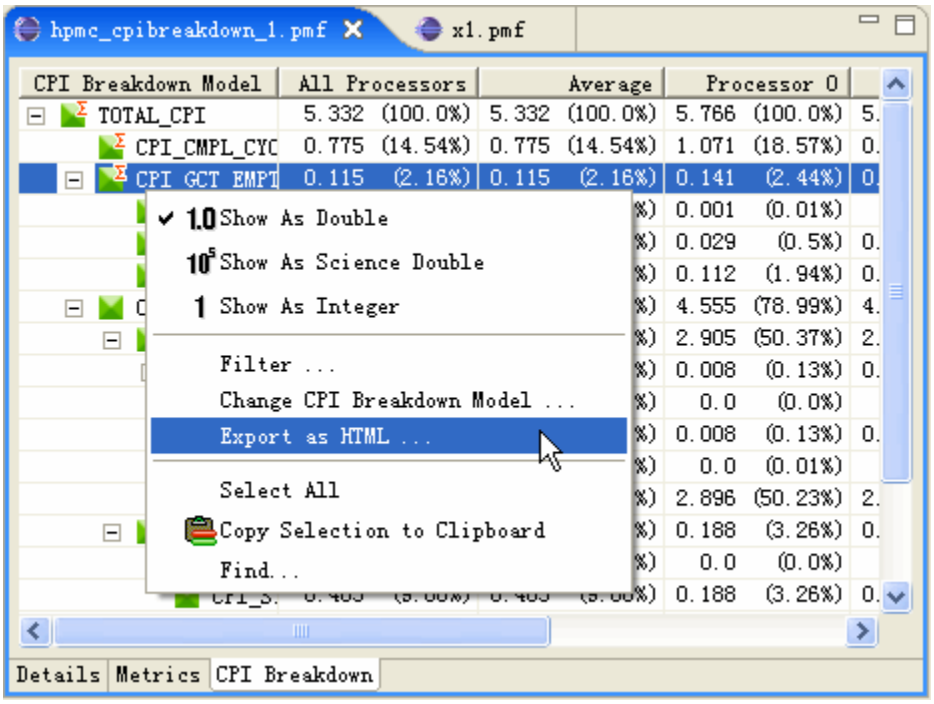


The CPI breakdown model to be selected can be either external files or built-in files. The Change CPI breakdown model dialog is as follows:

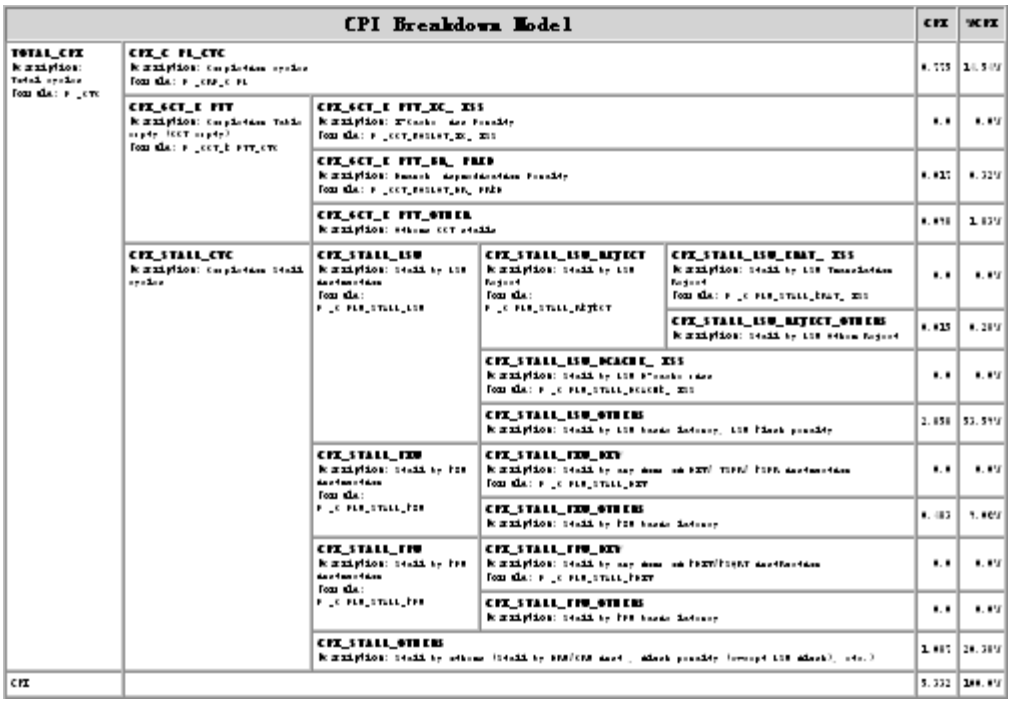


Export as HTML

Choose **Export as HTML** on the context menu, you can export CPI breakdown data into an HTML file.

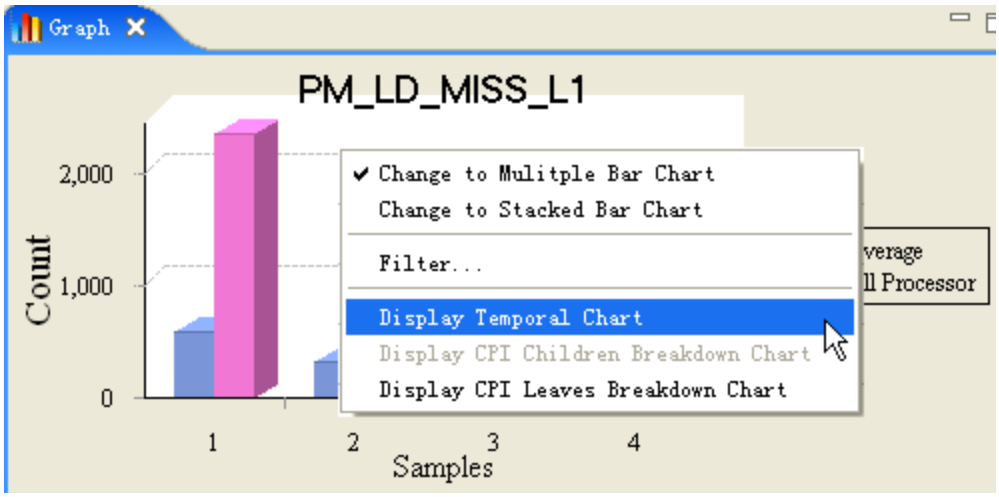


The CPI breakdown HTML file is as follows:

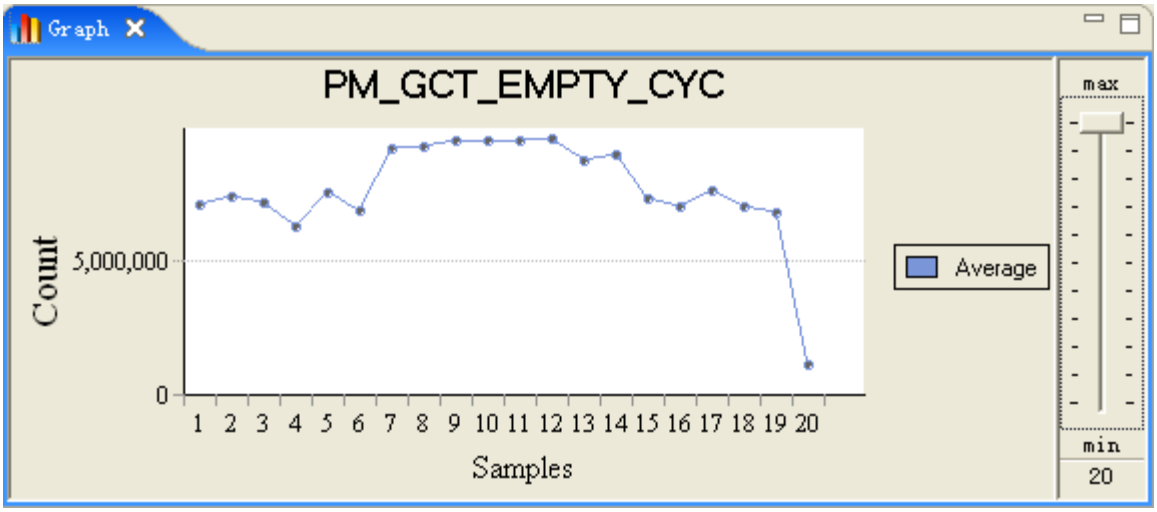


5.4.3.3View Temporal Chart

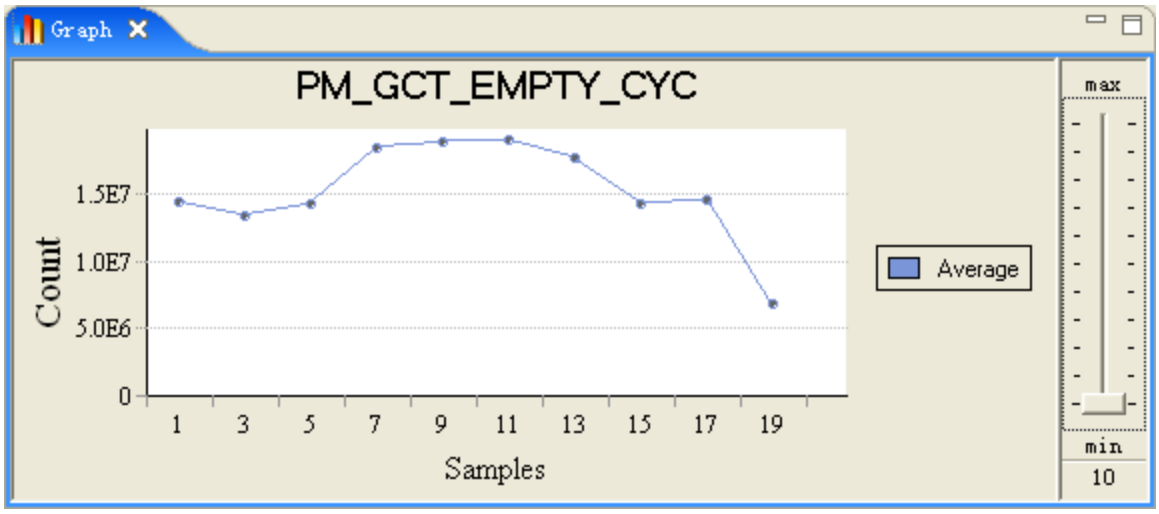
You can view temporal chart of the counter data in **Graph** view, by selecting **Display Temporal Chart** to enter temporal mode which is set as the default chart mode.



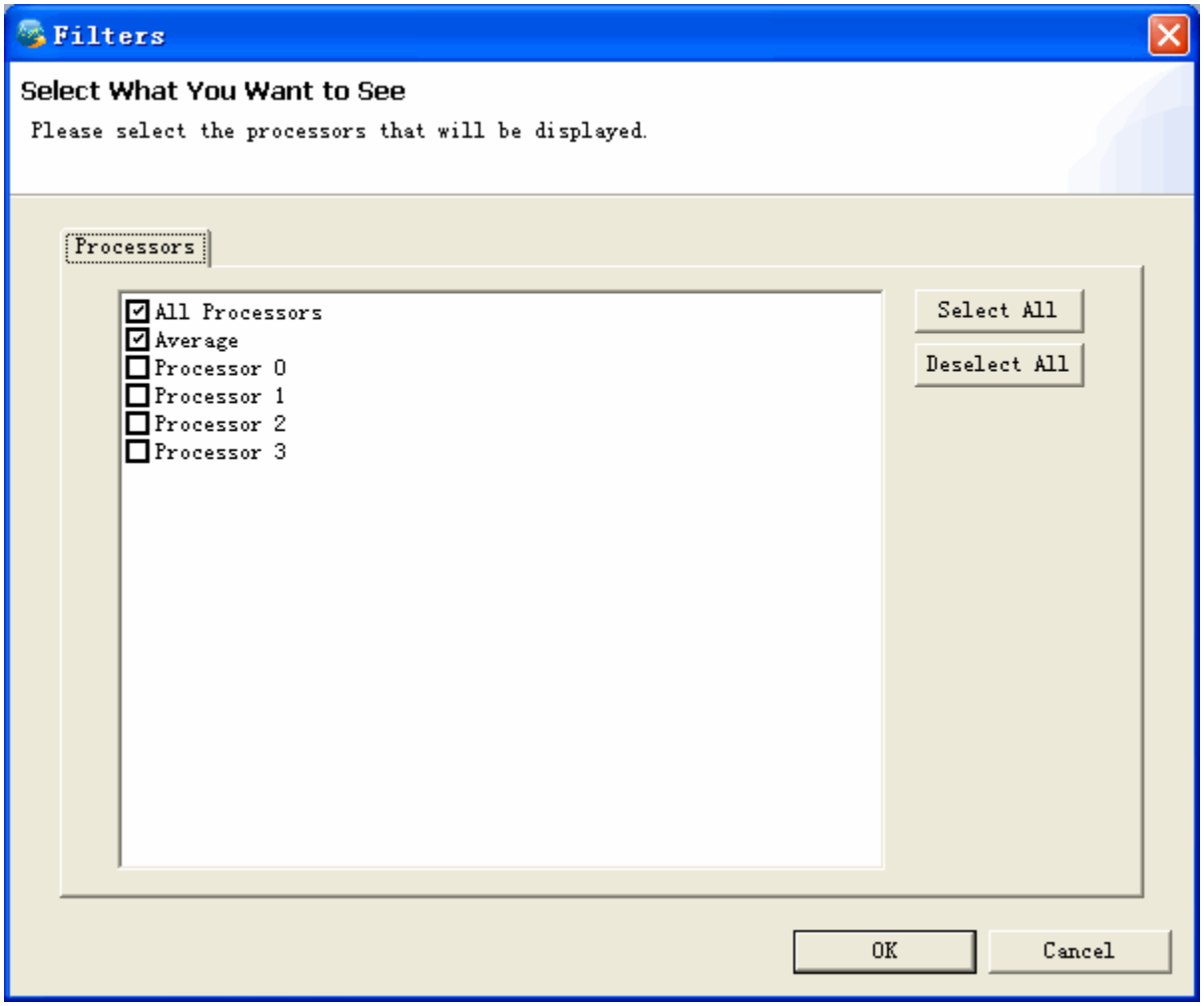
Aggregation Scale is used to scale the aggregation rate of samples. You may feel overwhelmed in front of too many samples in **Graph** View in temporal mode sometimes. And with the aggregation scale, it is easy for you to choose the number of samples to display. For example, you may choose to display 20 samples in **Graph** View:



And you may also choose to display 10 samples here:

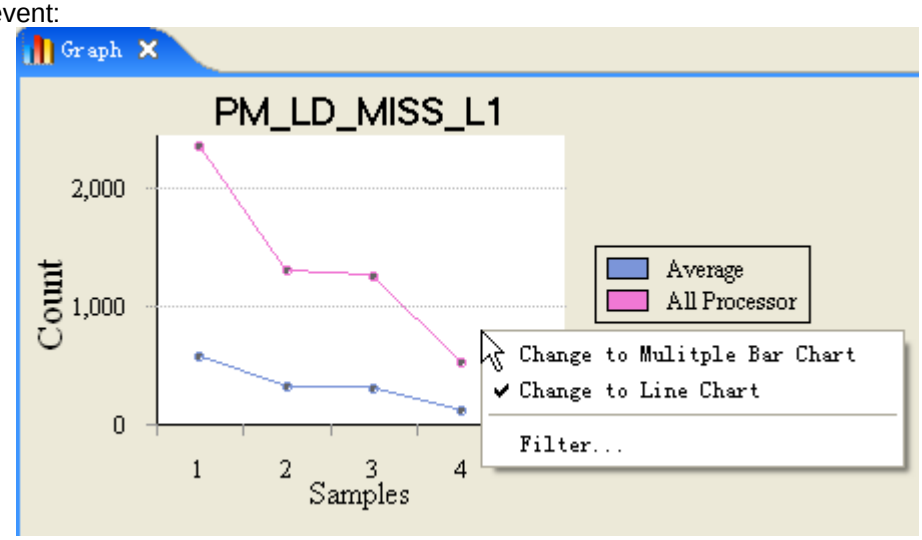


Filter is used to filter processors.

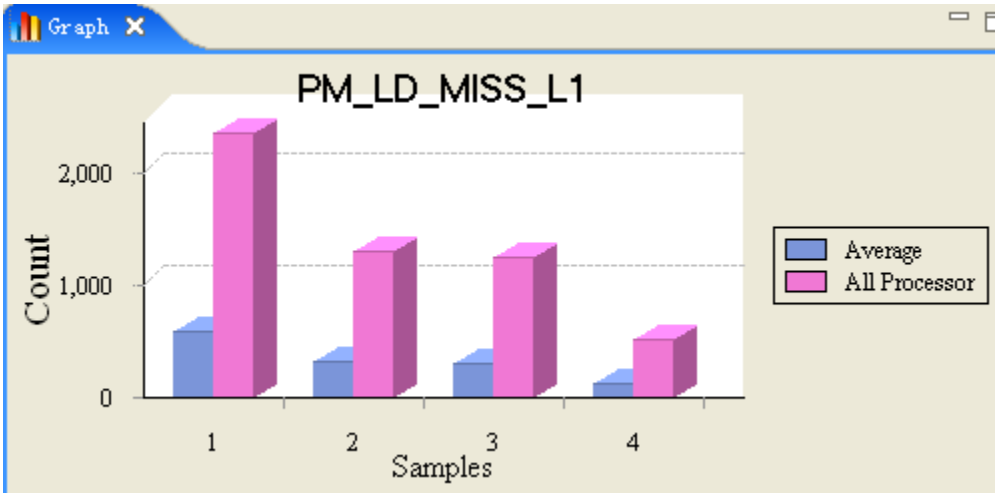


In temporal mode, when you select one event in Tab **Details**, one metric in Tab **Metrics**, or one CPI component in Tab **CPI Breakdown**, the temporal information of htis item is displayed in **Graph** view. The chart type can be switched between **Multiple Bar Chart** and **Line Chart**.

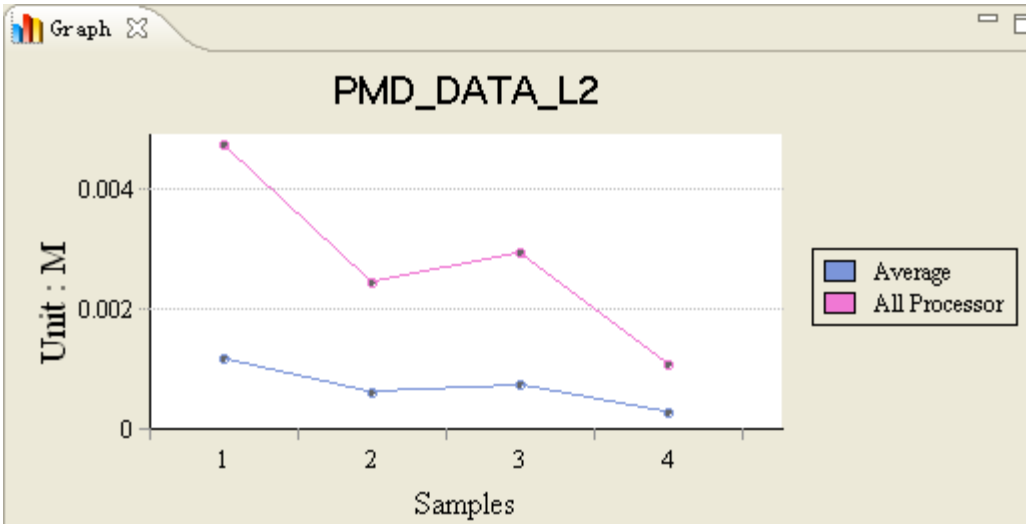
- Line chart of event:



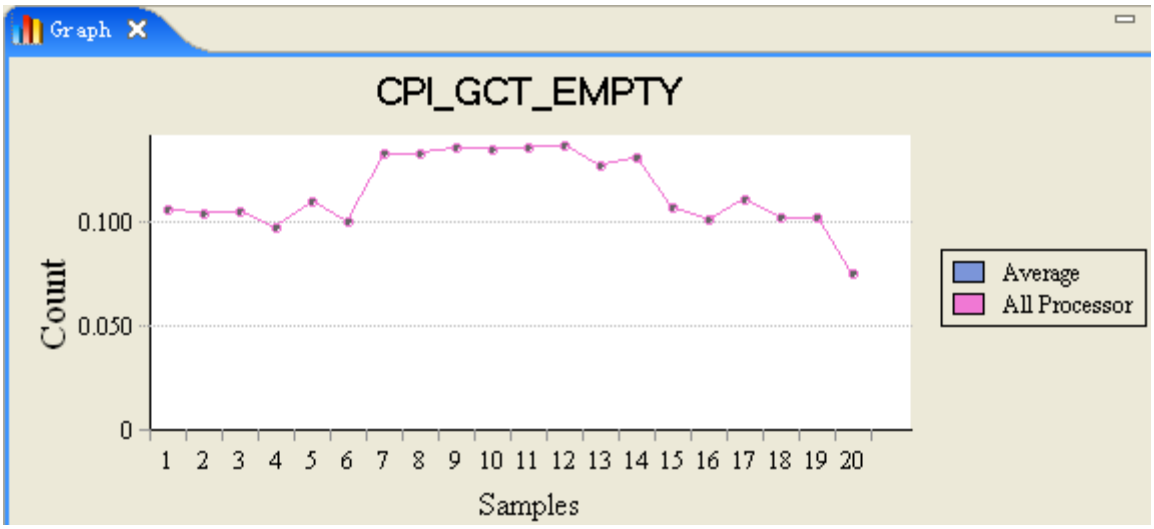
Multiple bar chart of one event:



- Line chart of one metric:



- Line chart of one CPI breakdown component:

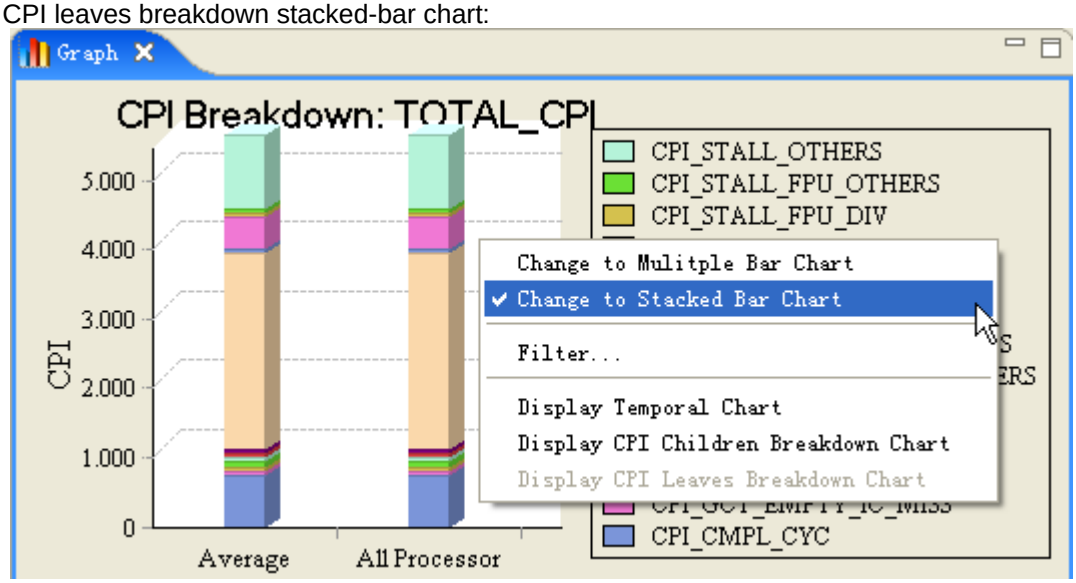


Note: This plot mode is supported by details data, metrics data, and CPI breakdown data.

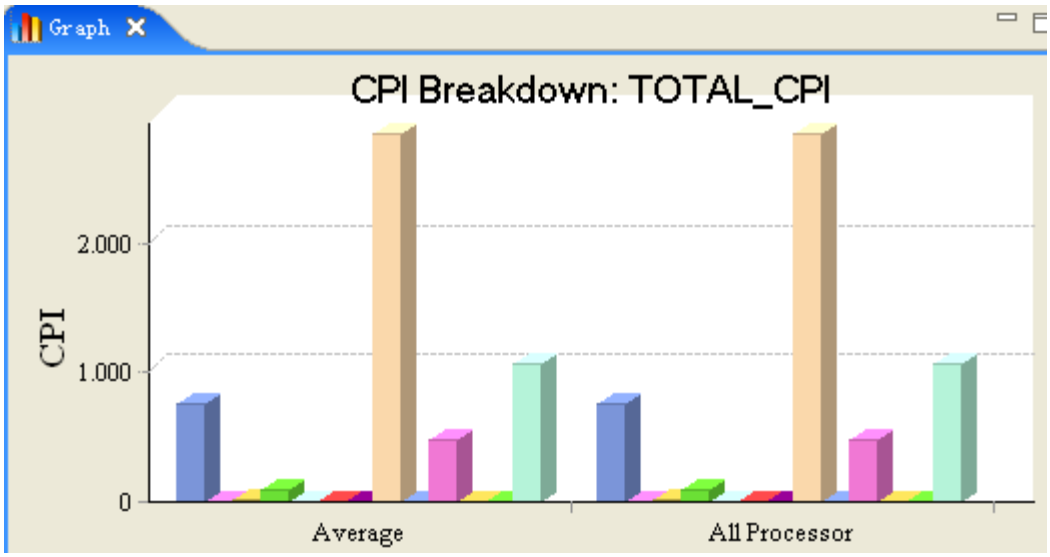
5.4.3.4View CPI Breakdown Chart

In CPI breakdown mode, when you select one CPI component in Tab **CPI Breakdown**, CPI breakdown data can be displayed in **Graph** View. You can choose to display **CPI Children Breakdown Chart** or **CPI Leaves Breakdown Chart**. The former merely displays the sons of the selected node, while the latter displays all the leaf nodes with the selected node as the root. In this mode, chart type can be switched between **Multiple Bar Chart** and **Stacked Bar Chart**. **Filter** is used to filter processors.

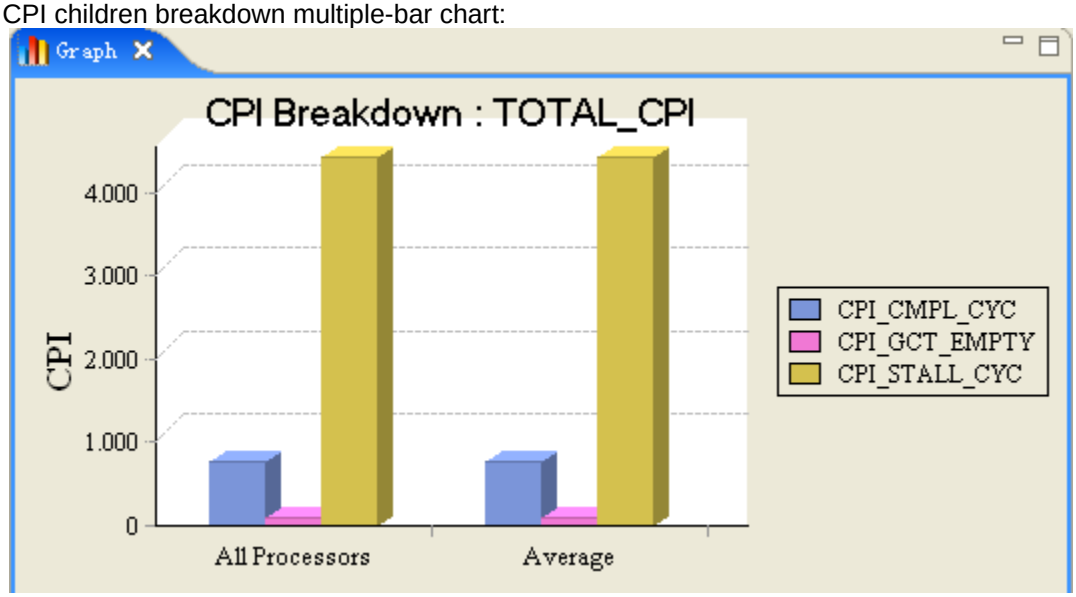
- Display CPI leaves breakdown stacked-bar chart:



- Display CPI children breakdown multiple-bar chart:



- Display CPI children breakdown multiple-bar chart:



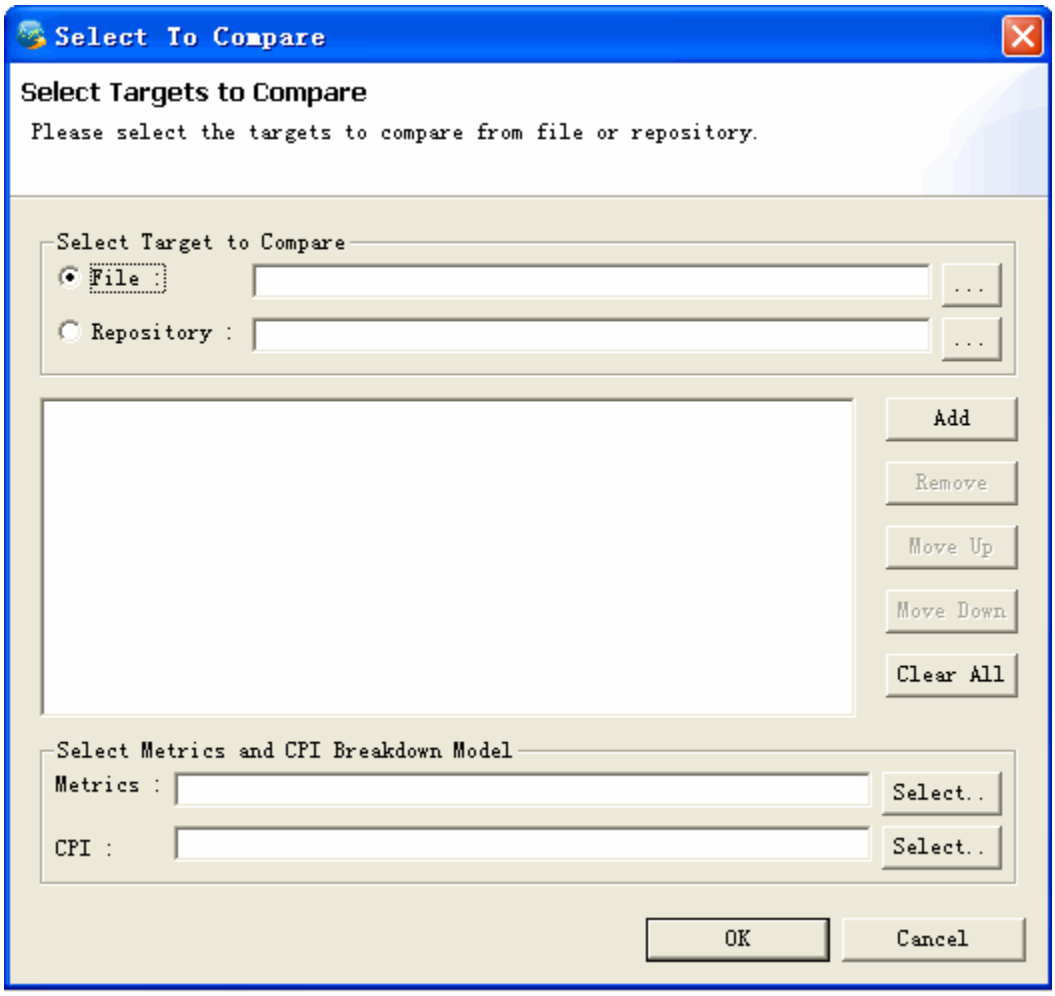
Note: This plot mode is only supported by CPI breakdown data.

5.4.3.5Compare Counter Data

You can click on the tool bar to select several files to compare. In the pop-up dialog, you must select 2~4 target files to compare. You can either select files from the explorer, or select files from repositories (you have to refresh repositories in **Counter Repository** View in davance). Select one file, and click the **Add** Button to add it. The first file you add is regarded as the base file.

You can further specify the metrics and CPI breakdown Model to apply to the counter data, or do it later (Please refer to **Change Metrics** and **Change CPI Breakdown Model**).

The following screen snapshot displays the **Select To Compare Dialog**:

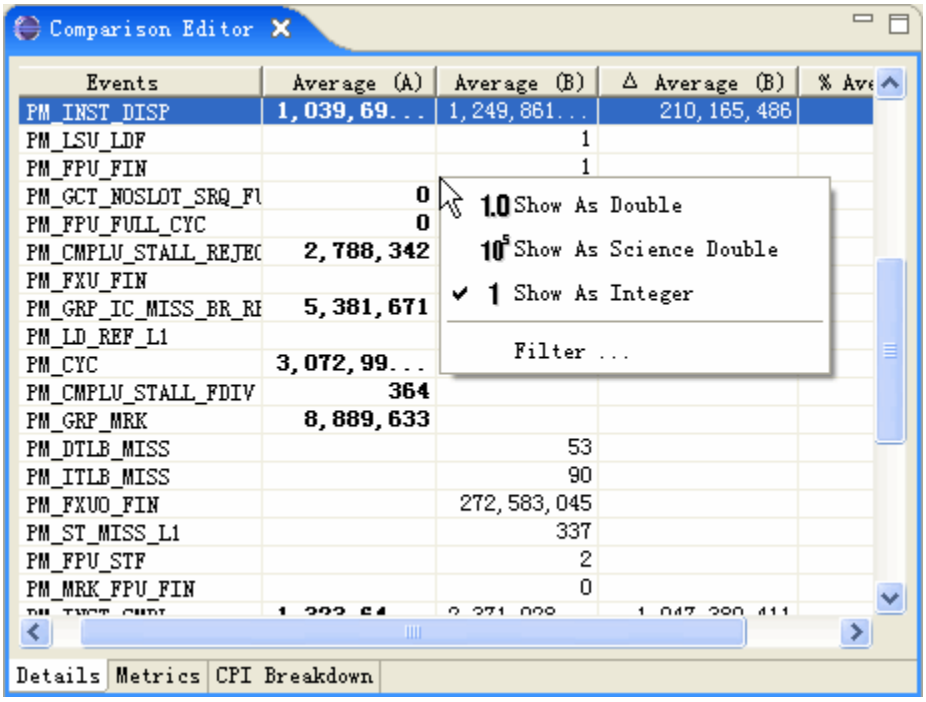


5.4.3.5.1View Comparison Data

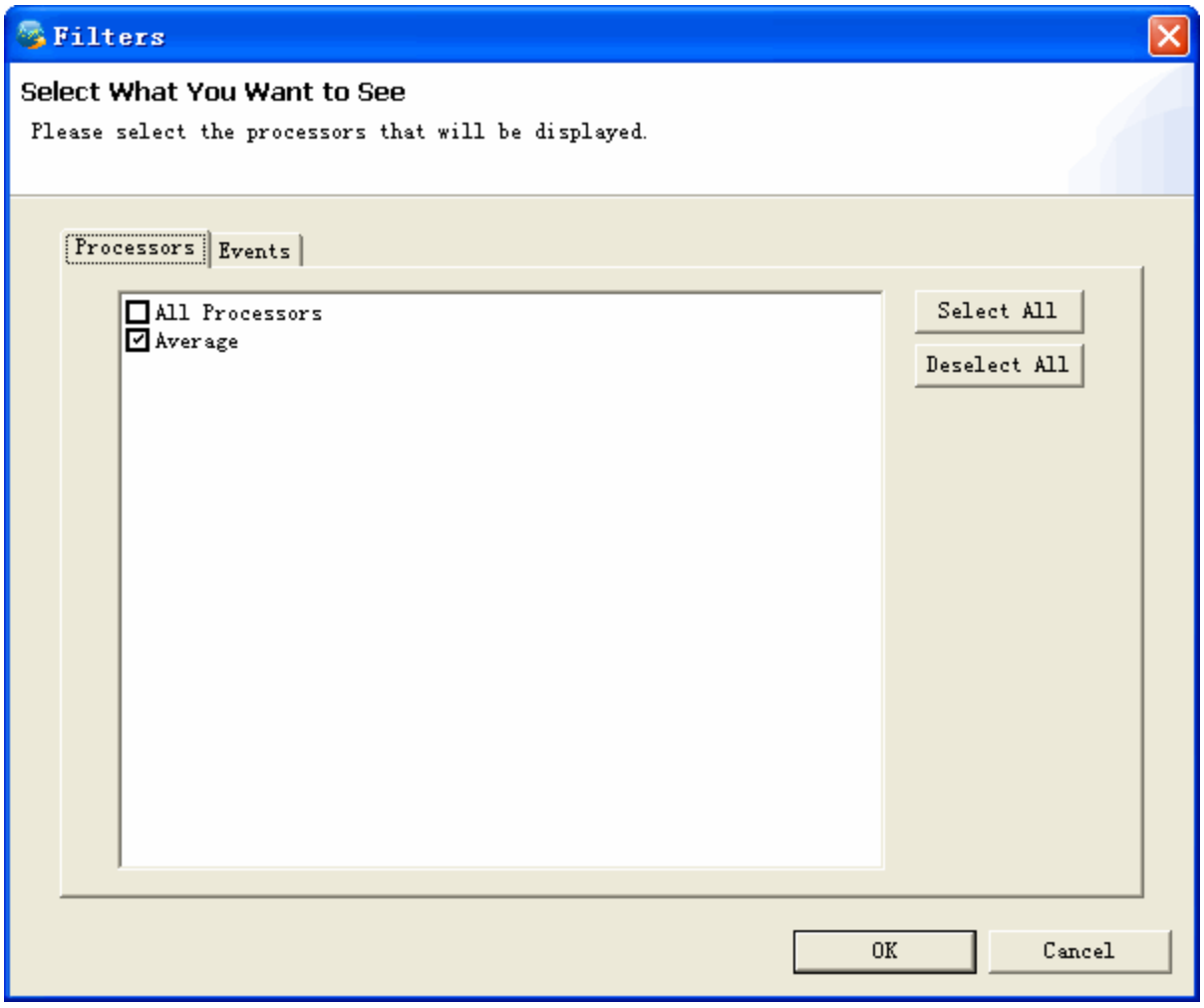
Comparison Editor is opened to display comparison result side by side. The following items can be compared:

- The total counts of one event are compared.
- The derived metrics values are compared.
- The CPI breakdown values are compared.

The following screen snapshot displays comparison data of each event from different data sources. refers to difference between the target file and the base file, while **%** refers to the target files' proportion of the base file.



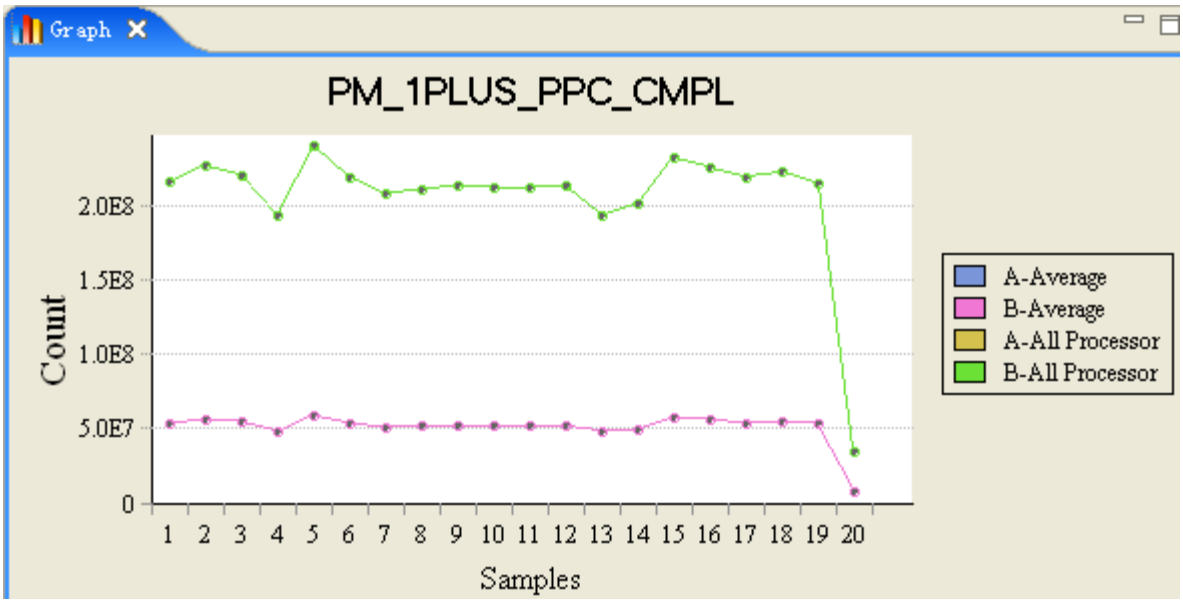
The comparison is based on the sum of all processors, the average of all processors, or both. You can specify it in **Filter**.



5.4.3.5.2View Temporal Comparison Chart

When Comparison Editor is opened, comparison data are displayed in **Graph** view. You can view temporal chart of the comparison data by selecting **Display Temporal Chart** to enter temporal mode which is set as the default chart mode.

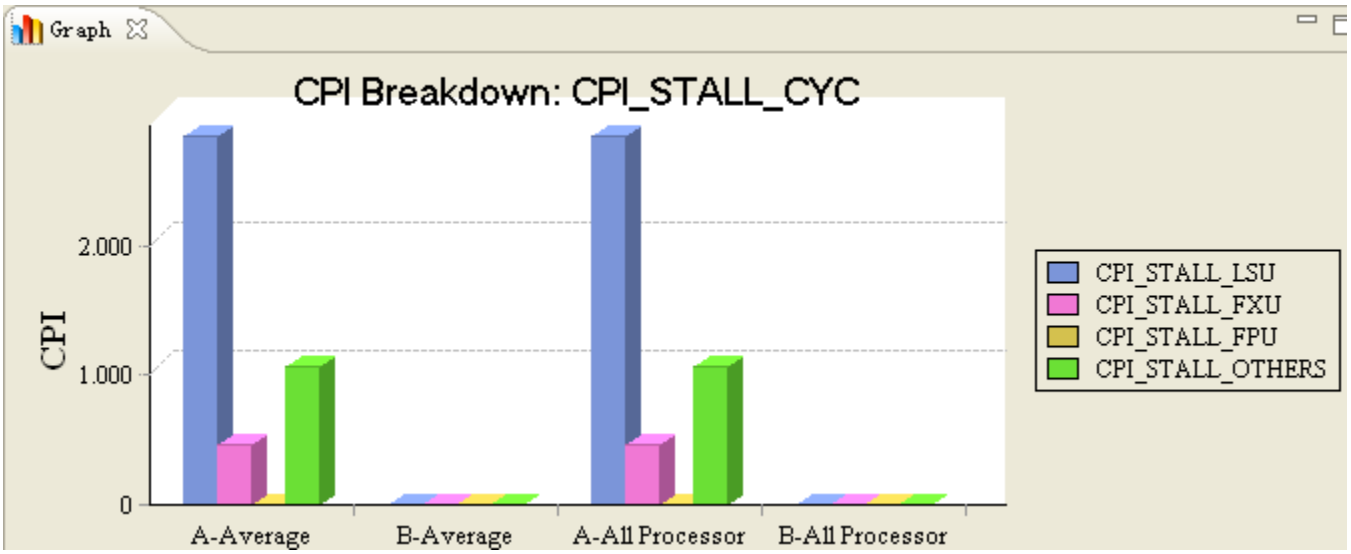
The following screen snapshot displays the temporal comparison line chart of one event. The temporal comparison chart of one metric and one CPI breakdown component is much the same with that of one event. The comparison is based on the sum of all processors, the average of all processors, or both. You can specify it in **Filter**.



Note: This plot mode is supported by details data, metrics data, and CPI breakdown data.

5.4.3.5.3View CPI Breakdown Comparison Chart

When Comparison Editor is opened, comparison data are displayed in **Graph** view. When you select one CPI component in Tab **CPI Breakdown**, you can choose to display **CPI Children Breakdown Chart** or **CPI Leaves Breakdown Chart**. The former merely displays the sons of the selected node, while the latter displays all the leaf nodes with the selected node as the root. In this mode, chart type can be switched between **Multiple Bar Chart** and **Stacked Bar Chart**. The comparison is based on the sum of all processors, the average of all processors, or both. You can specify it in **Filter**.

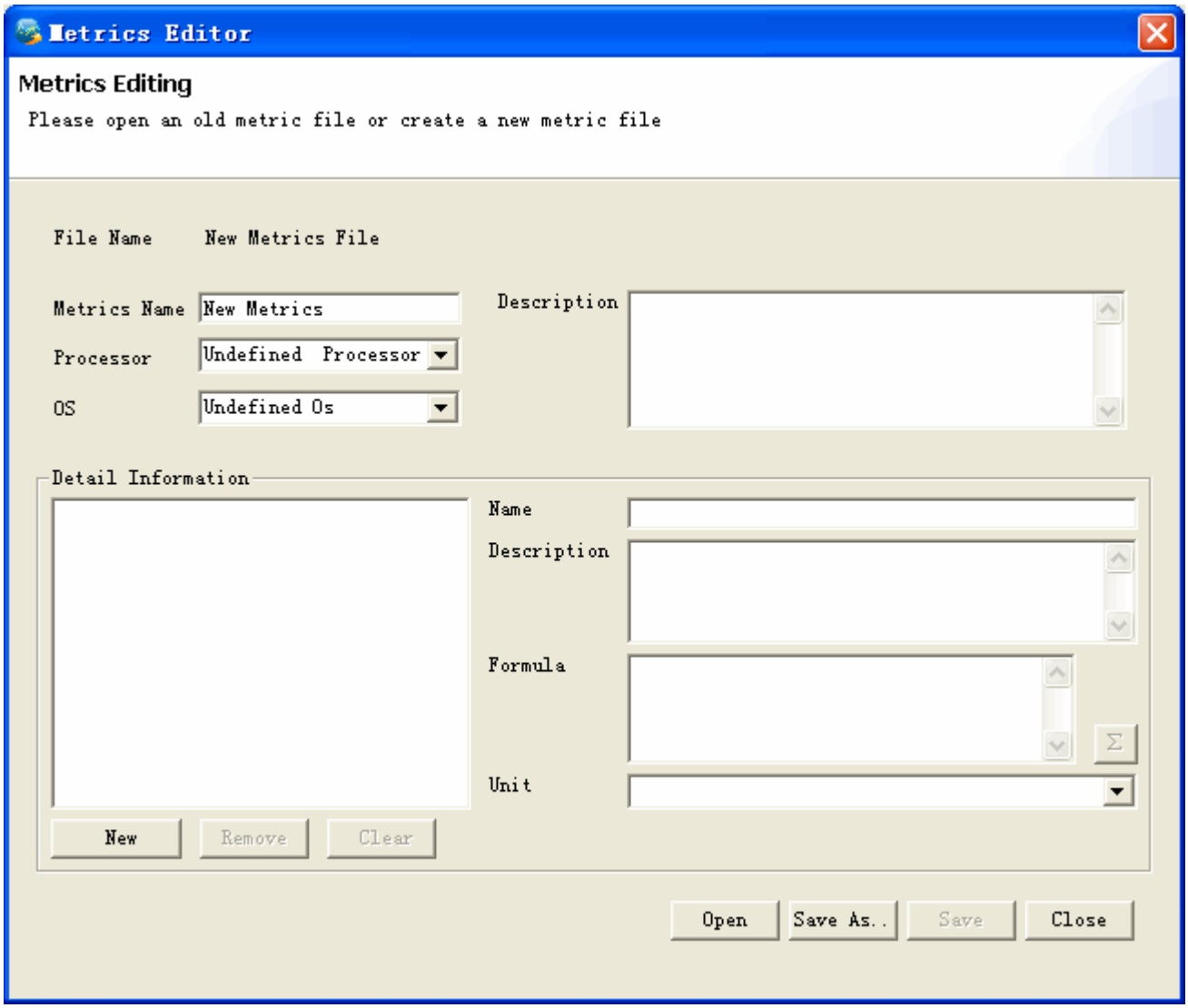


Note: This plot mode is only supported by CPI breakdown data.

5.4.3.6Edit Metrics/CPI Breakdown Model

5.4.3.6.1Edit Derived Metrics

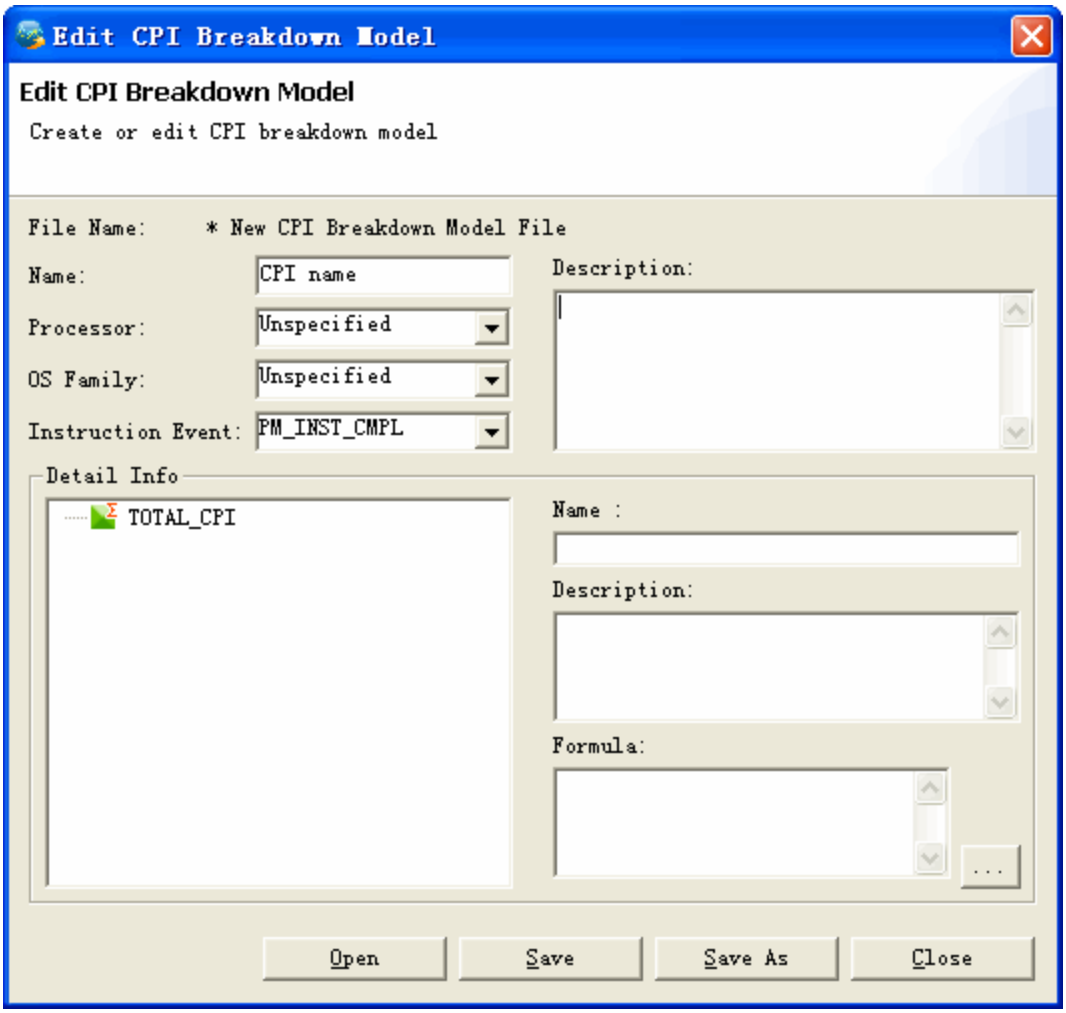
1. You can select on the tool bar to open one derived metrics file to edit.
2. You can view and edit derived metric formula in **Metrics Editing** dialog. The supported operation includes:
   - Add new derived metric
   - Edit derived metric
   - Delete derived metric



3. After modification, you can save the file or save it as another derived metrics file.

5.4.3.6.2Edit CPI Breakdown Model

1. You can select on the tool bar to open one CPI breakdown model definition file.
2. You can view and edit CPI breakdown model in **Edit CPI Breakdown Model** dialog. The supported operation includes:
   - Add new component
   - Edit component
   - Delete component



3. After modification and validation, you can save the file or save it as another CPI breakdown model definition file.

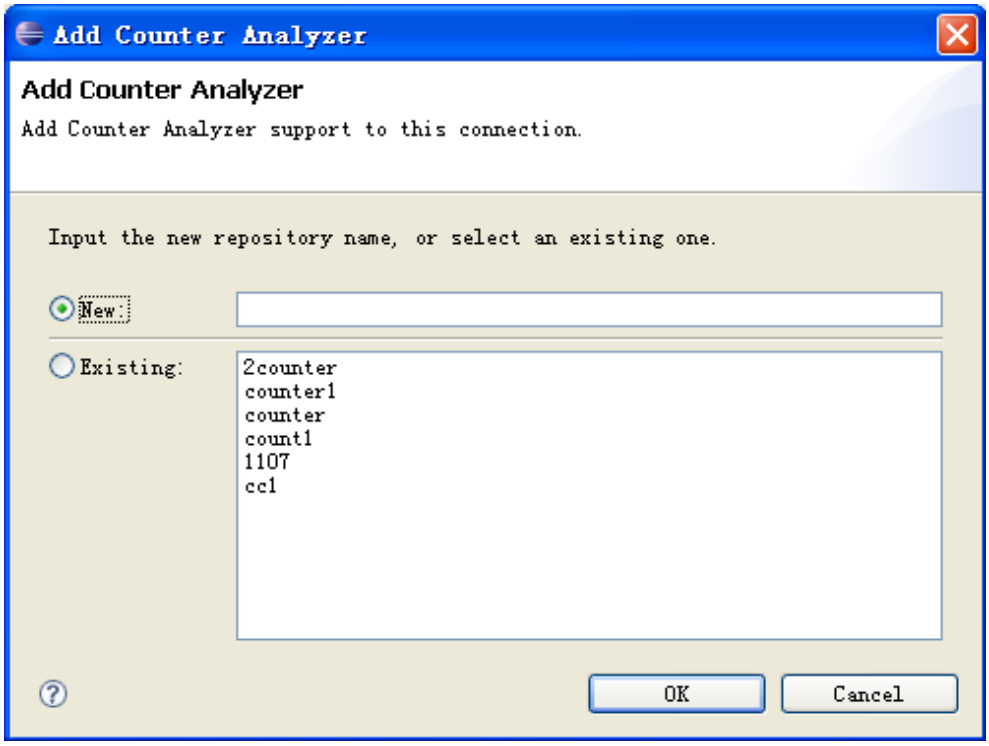
5.4.3.7 Import Counter Data File into Repository

To create Counter Analyzer Support, you should create a database connection first. Please refer to section 5.1.7 for details on how to create and manage database connections.

Then, right-click on the connection to create Counter Analyzer Support,

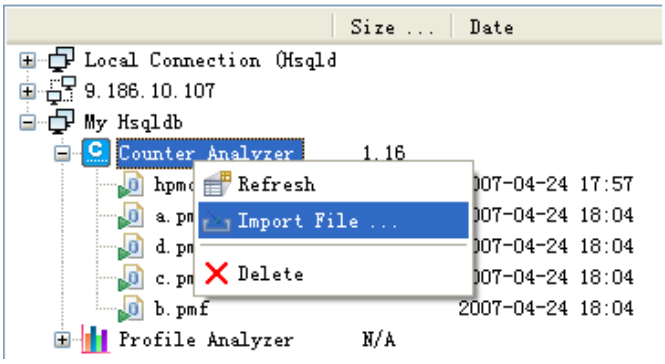
If you are using hsqldb connection, all existing database under this path will be listed, you can create a new one, or attach an existing one.

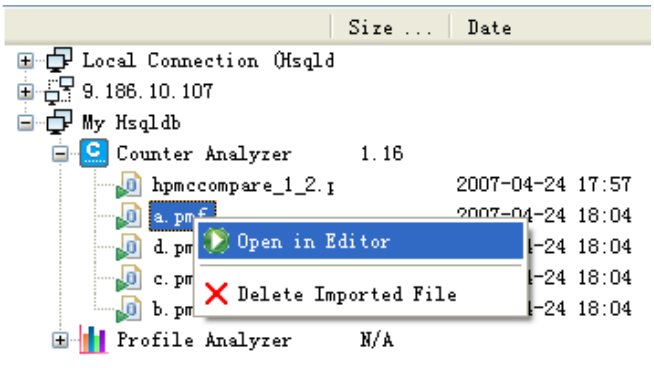
If you are using db2 connection, all existing schema under this database will be listed, you can create a new one, or attach an existing one.



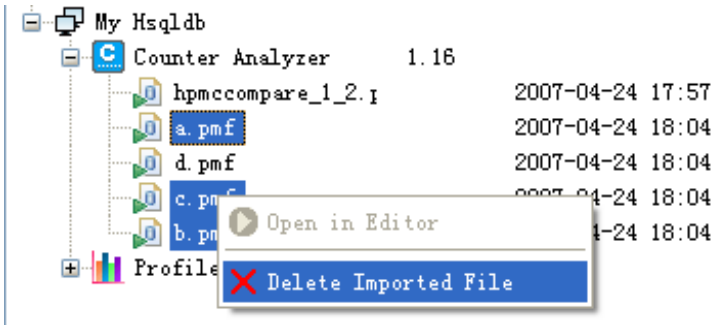
Create Counter Analyzer Support

After Counter Analyzer Support has been created, then, you can import file into this support, open it in editor, or delete it.

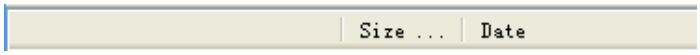




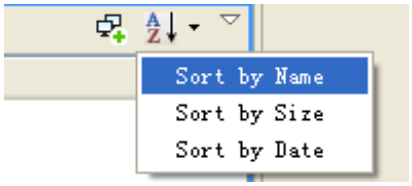
You can delete files manually to release disk space. Multi-selecting is allowed. See the picture below.



This view support sorting operation. You can sort the files by name, size or date.



To sort files, select the sort mode on the action bar, or click the title directly.



5.5Trace Analyzer

Trace Analyzer visualizes Cell/B.E. traces containing information such as DMA communication, locking/unlocking activities, mailbox messages, etc.

Trace Analyzer provides several views that help the user make sense of the trace data. The trace can be plotted in a graphical view, organized by core, along a common timeline. Alternatively, the user can traverse the trace records in a textual table, Another view provides the detailed data for each kind of records, for example, lock identifier for lock operations, accessed address for DMA transfers, etc.

You can also find the Trace Analyzer User Guides from the VPA. Select **Help - Help Contents** within VPA. To get context sensitive help, press **F1** for Windows and AIX or press **Ctrl+F1** for Linux.

5.5.1Basic concepts

- **Events**

Events are records that have no duration, for example, records describing non-stalling operations, such as releasing a lock. Events' input on performance is normally insignificant, but they may be important for understanding the application and tracking down sources of performance problems.

- **Intervals**

Intervals are records that may have non-zero duration. They normally come from stalling operations, such as acquiring a lock. Intervals are often a very significant performance factor, and identifying long stalls and their

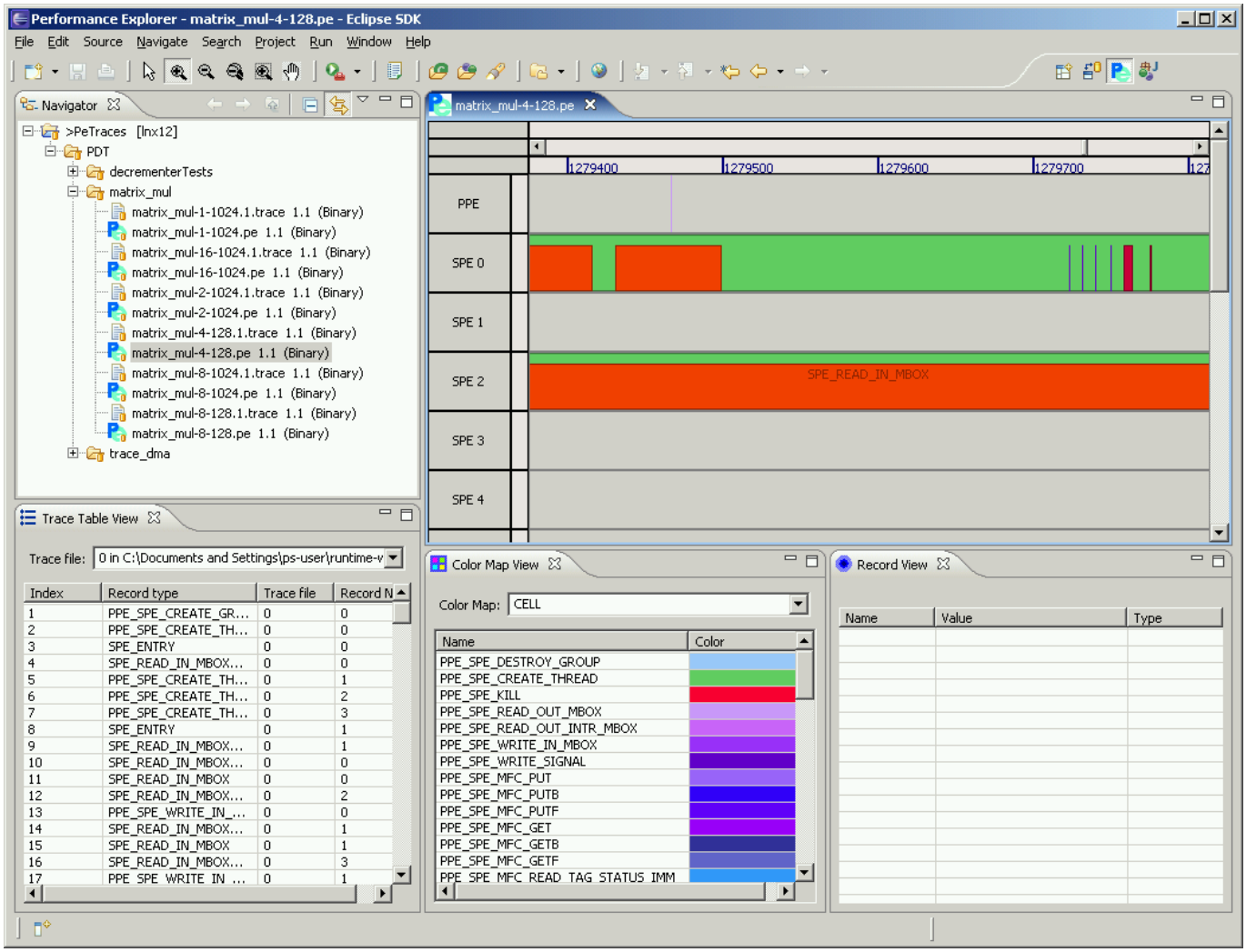
sources is an important task in performance debugging. A special case of an interval is *live interval*, that starts when an SPE thread begins to execute and ends when the thread exits.

5.5.2Load an existing trace file

5.5.2.1Open Trace Analyzer Perspective

When you first start Visual Performance Analyzer after installation, the default perspective is Profile Analyzer. To start Trace Analyzer, choose **Window -> Open Perspective -> Other.** In the Select Perspective dialog choose **Trace Analyzer** and click **OK**.

The following screen capture shows the initial layout of Trace Analyzer.



5.5.2.2Load in trace file

Choose **File -> Open File**_, and in Open File dialog select one trace file with suffix ".pe"..

5.5.2.3Brief introduction to Trace Analyzer Perspective

After loading in the trace data, the Trace Analyzer Perspective displays the data in its views and editors. Going from the top left clockwise, we see:

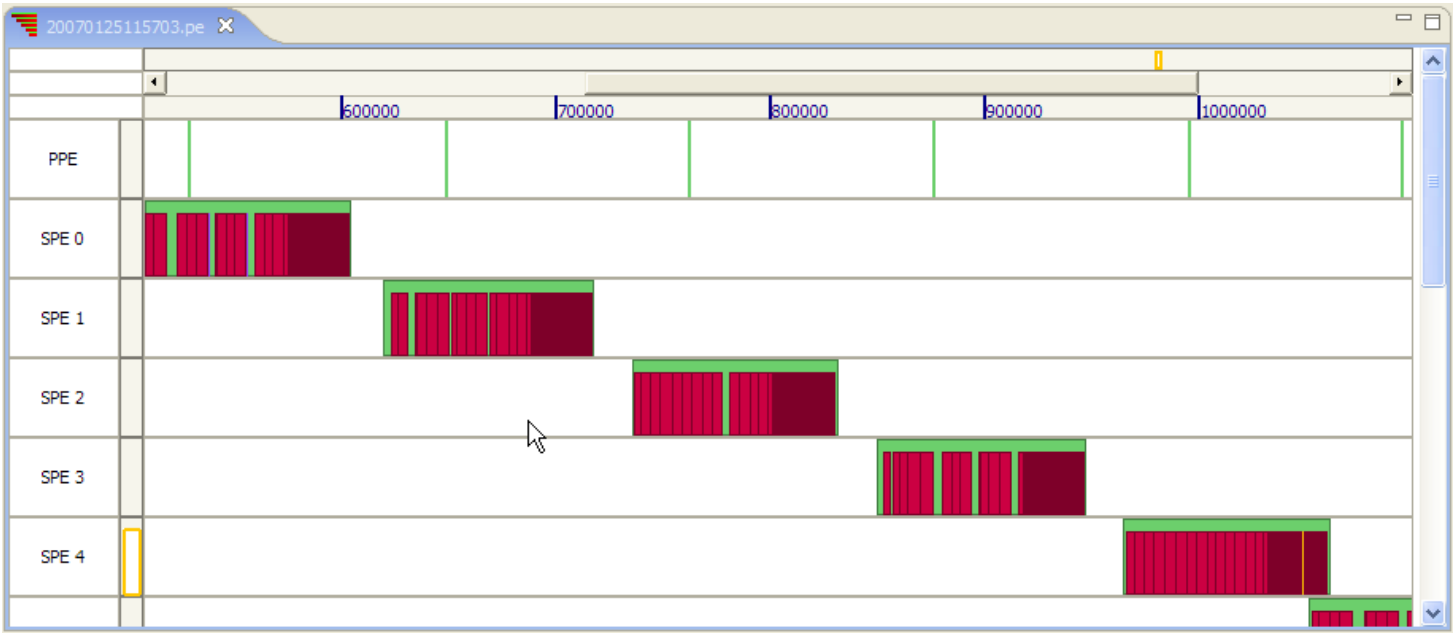
- **Navigator View**
- **Trace Editor** showing the trace visualization by core
- **Details View** showing the details of the selected record (if any)
- **Color Map View**, allowing the user to view and modify color mapping for different kinds of events
- **Trace Table View**, which shows all the events on the trace in the order of their occurrence

5.5.3Navigate the Trace Analyzer Perspective

The following are tasks that you can perform to navigate around trace data within Trace Analyzer.

5.5.3.1View Trace Data Graph

A graphical presentation of the trace is shown in the editor window.

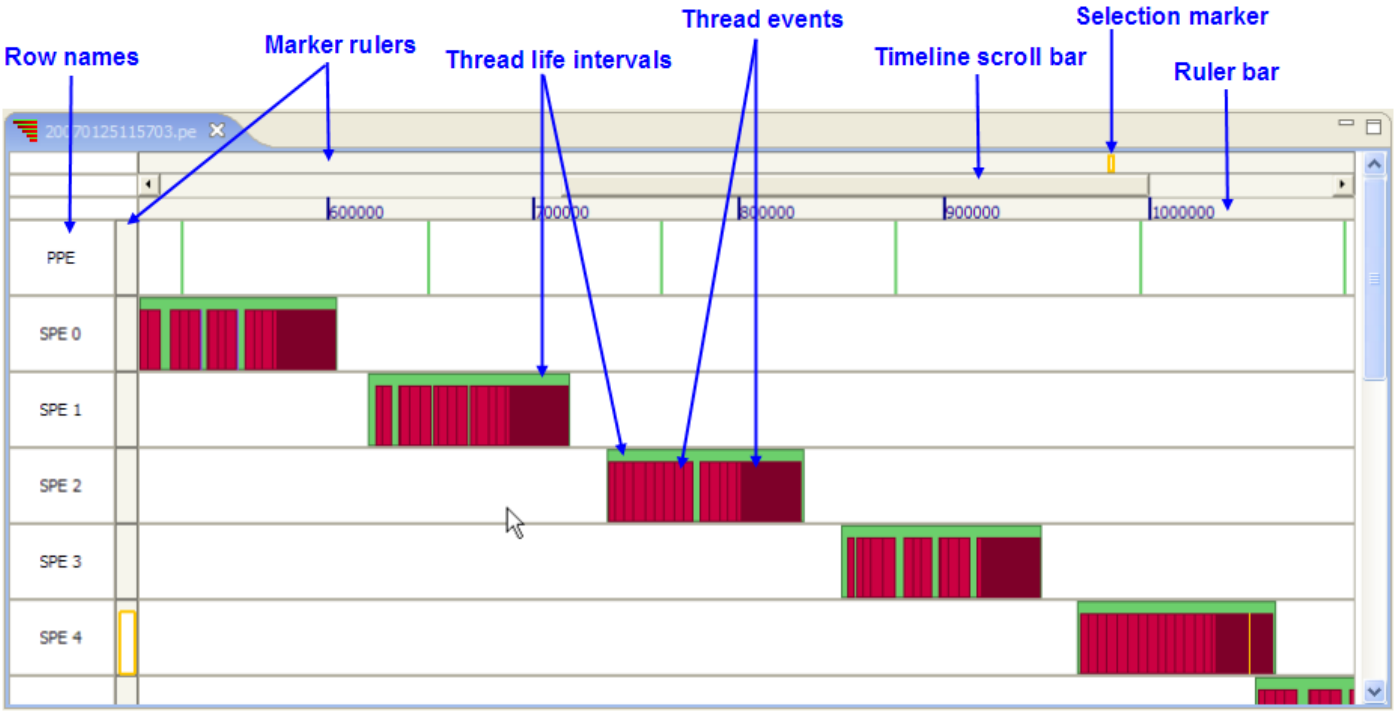


Data from each processor is displayed in a separate row, and each trace record is represented by a rectangle. Time is represented on the horizontal axis, so that the location and size of a rectangle on the horizontal axis represent the corresponding event's time and duration. The color of the rectangle represents the type of event, as defined by the Color Map View.

In the rows corresponding to the SPEs, note the full-height green rectangles. They show the live intervals, i.e., the life time of SPE threads. Records represented by the shorter reddish rectangles represent events that occurred during the thread execution.

Trace Editor Components

The following figure shows the different components of the Graphical View.



Going from top to bottom, we have:

- The *marker ruler* shows where the selected record (if any) is located on the trace's timeline (look for the orange-and-white selection marker). Clicking on the marker ruler scrolls the view to make the selected event visible. Note that there are also vertical marker rulers, located in each row between the core id and the graph. These rulers show on which core the selected event occured.
- The *scrollbar* can be used to scroll back and forth in time.
- The *ruler bar* shows the time values, in the same units as those used in the trace.

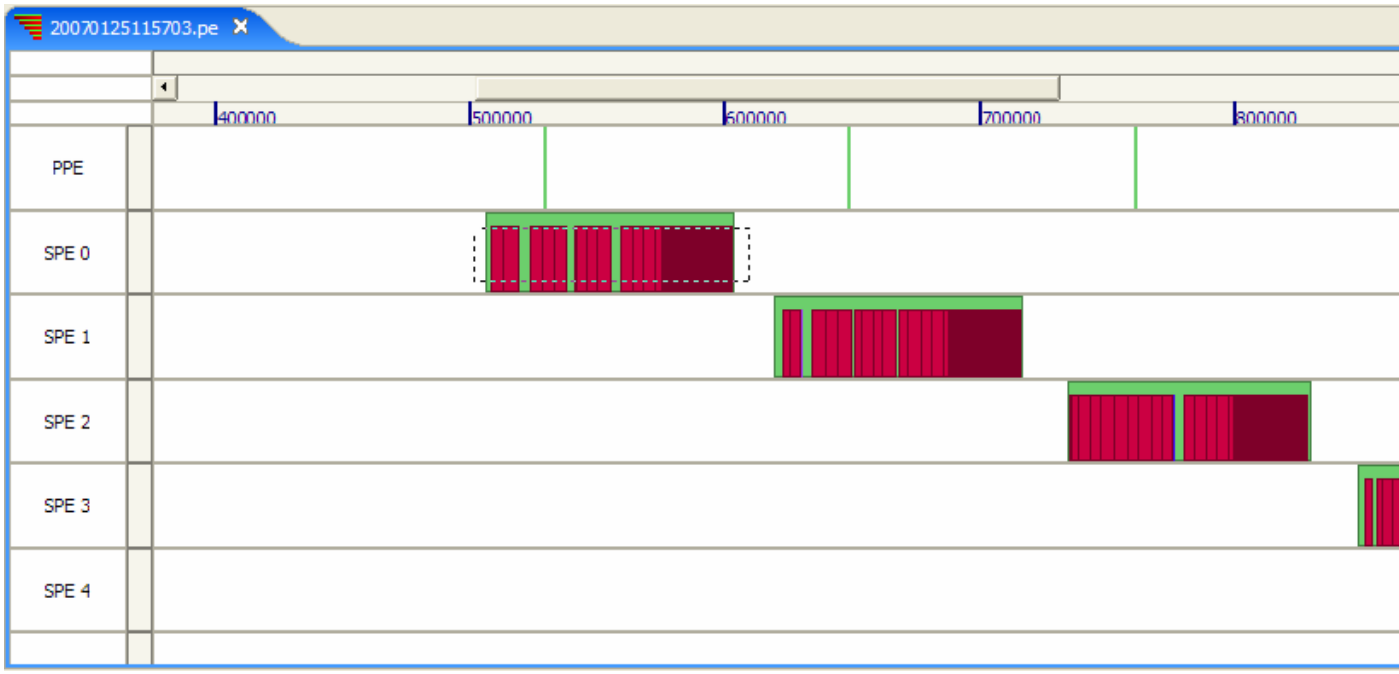
Trace Editor Tools

When a trace is open in the Graphical View, the following toolbar is added to the standard Eclipse toolbars:

This toolbar is only active when the focus is on the Trace Editor. The following tools are available:

-  : *Selection tool*. Pick this tool and click with it a record on the Graphical View to select the record. This will scroll the Record List View to the selected record and display its details in the Record Details View.

- : *Zoom-in point tool*. Pick this tool and click one of the graphs to zoom in while keeping the time value at the click point at the same location.
- : *Zoom-out point tool*. Pick this tool and click one of the graphs to zoom out while keeping the time value at the click point at the same location.
- : *Zoom-all tool*. Pick this tool and click anywhere in the graph to fit all the trace into the view.
- : *Zoom-in area tool*. To fit a specific region into the view, pick this tool and in one of the rows mark the area you want to fit into the view:

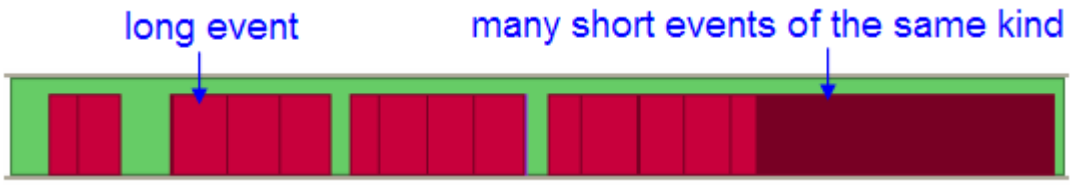


- : *Drag tool*. To scroll the view back and forth along the time axis, pick this tool, and hold the right mouse button pressed while dragging the graph

Trace Editor Coloring Conventions

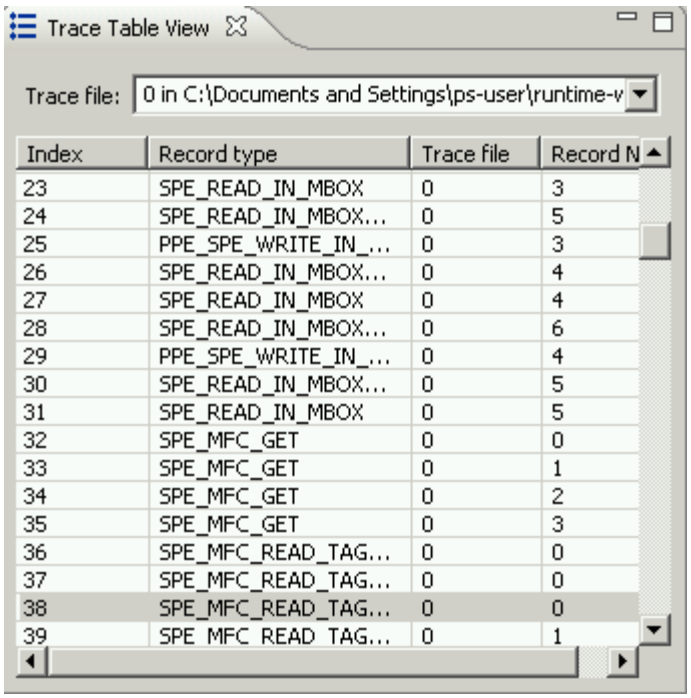
The default coloring used by the Trace Editor assigns hues of blue to events and hues of red to intervals. One of the few exceptions are the live intervals, colored green. Please refer to the Color Map View for the exact color mapping of any particular editor instance.

When analyzing a trace, it is often important to distinguish between a large number of short intervals and a single long interval, which may be a good target for optimization. In order to aid in this analysis, Trace Analyzer emphasizes intervals with a border whose color is a darker hue of the event's color:



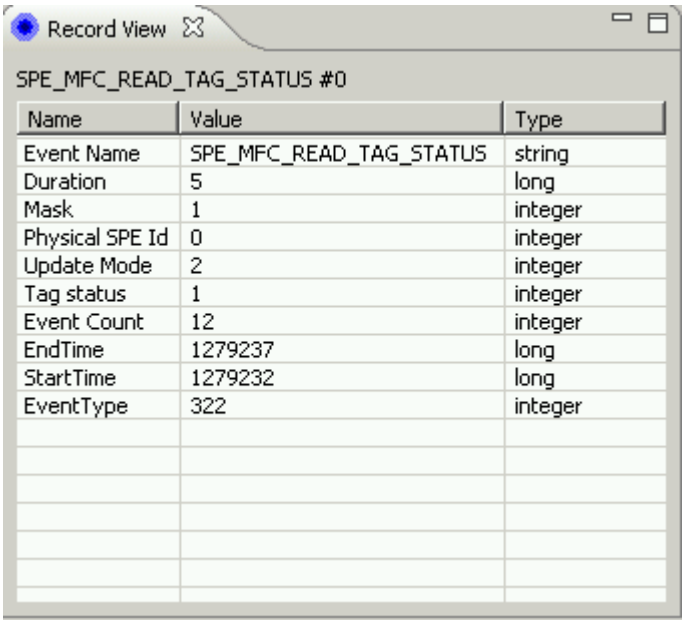
5.5.3.2View List of Trace Records

You can view the list of the records in the trace in the **Trace Table View**. Click on a row to select a record and see its details in the **Record Details View**. **Trace Table View** selection is also synchronized with the selection in the **Trace Editor**, so that each scrolls to and highlights the selection done in the other.



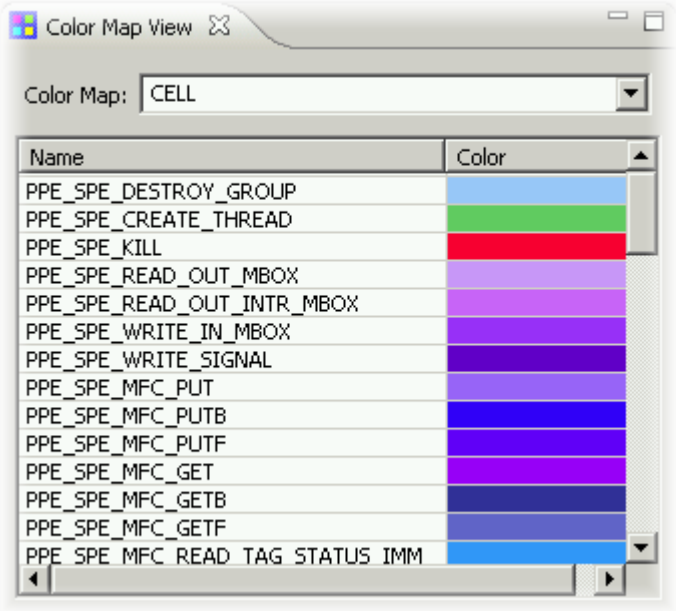
5.5.3.3View Record Details

The **Record Details View** shows the names and values for all the fields defined for the selected record.



5.5.3.4Change Colors used in Trace Editor

The **Color Map View** holds a list of record types and their color mapping. To change the color mapping for a particular record type, double-click the corresponding row. A color chooser dialog will open, allowing to select the new color for the record type.

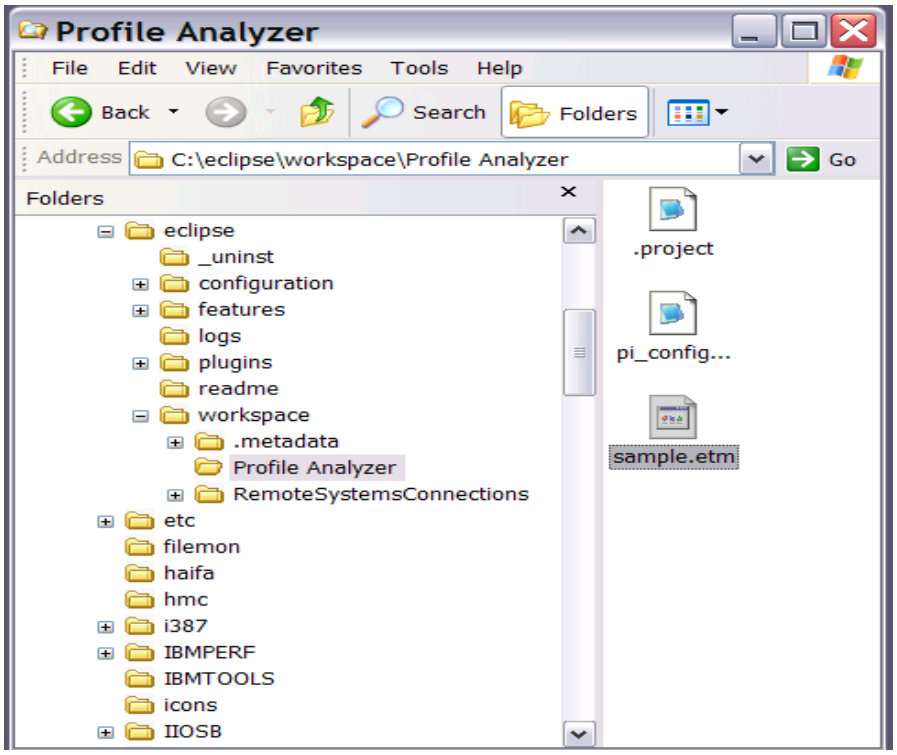


Note that the interval types, which are emphasized with a darker border in the Trace Editor, are also shown with a border in the Color Map View:

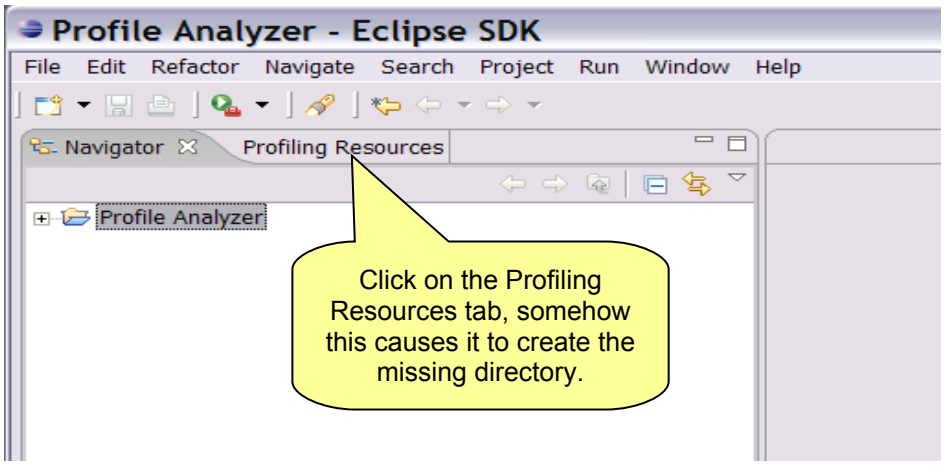
6.Appendix A - sample profiling session

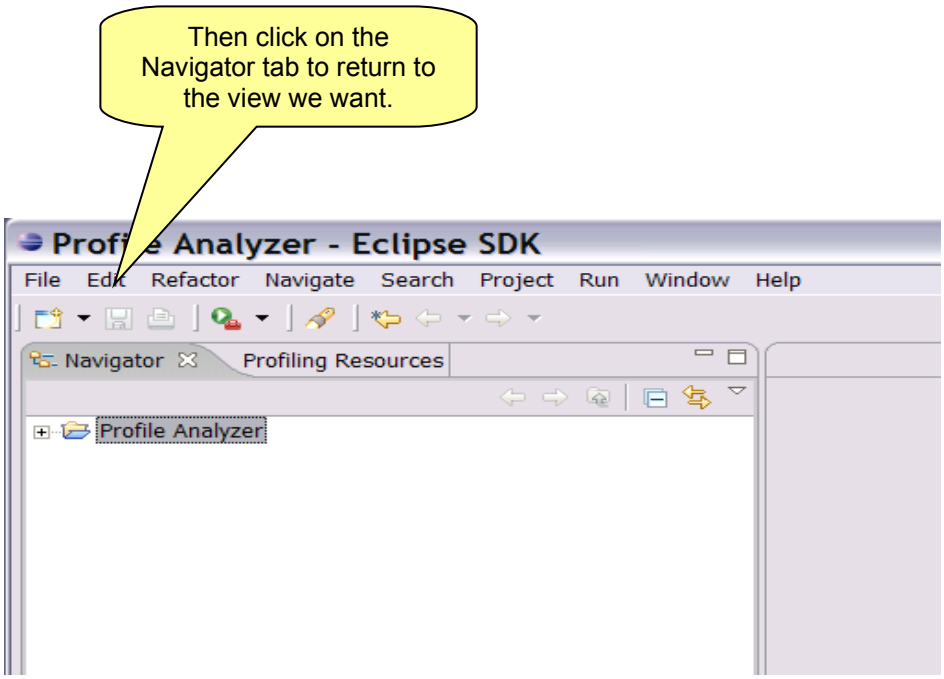
This process will walk through the typical usage of VPA to analyze problems, using the **Profile Analyzer plug-in** and some existing data. Where can you download the existing data? Look for the ***Sample ETM File*** at the bottom of the web page: http://www.alphaworks.ibm.com/tech/vpa/download.

The picture below shows the file being saved to a workspace directory.

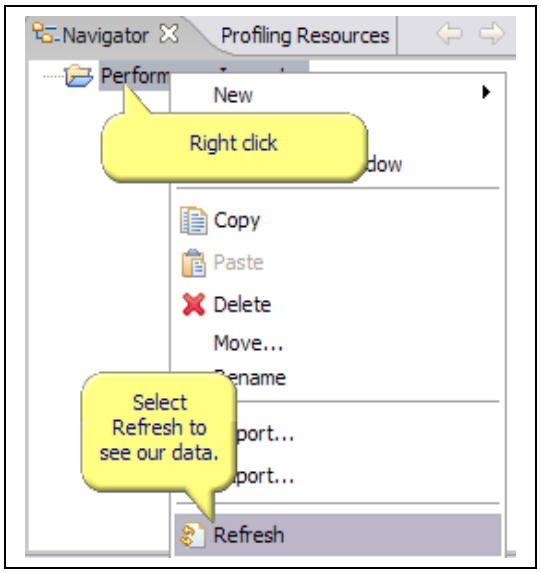


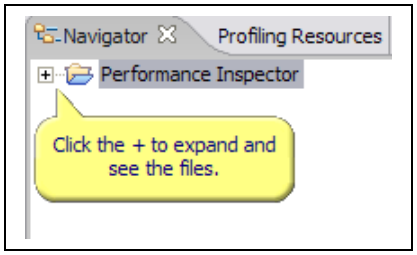
If you don't have the Profile Analyzer directory, try this:





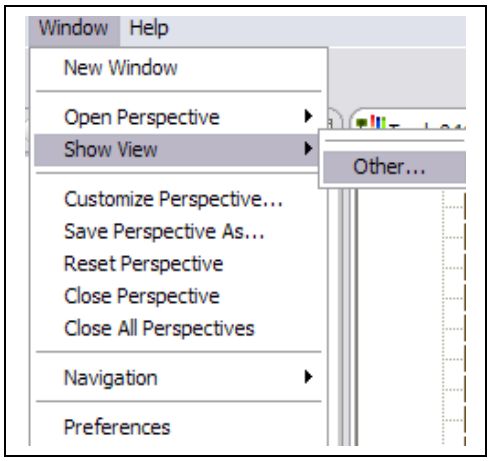
Once we have the Profile Analyzer folder in view, it is sometimes necessary to REFRESH it. Just Right click and select REFRESH.



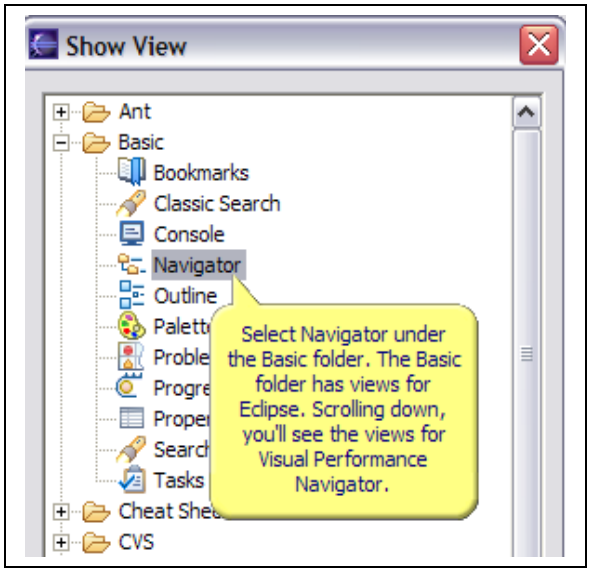


If you don't have a Navigator tab, adding a new view is easy with VPA!

Just select Window => Show View => Other…

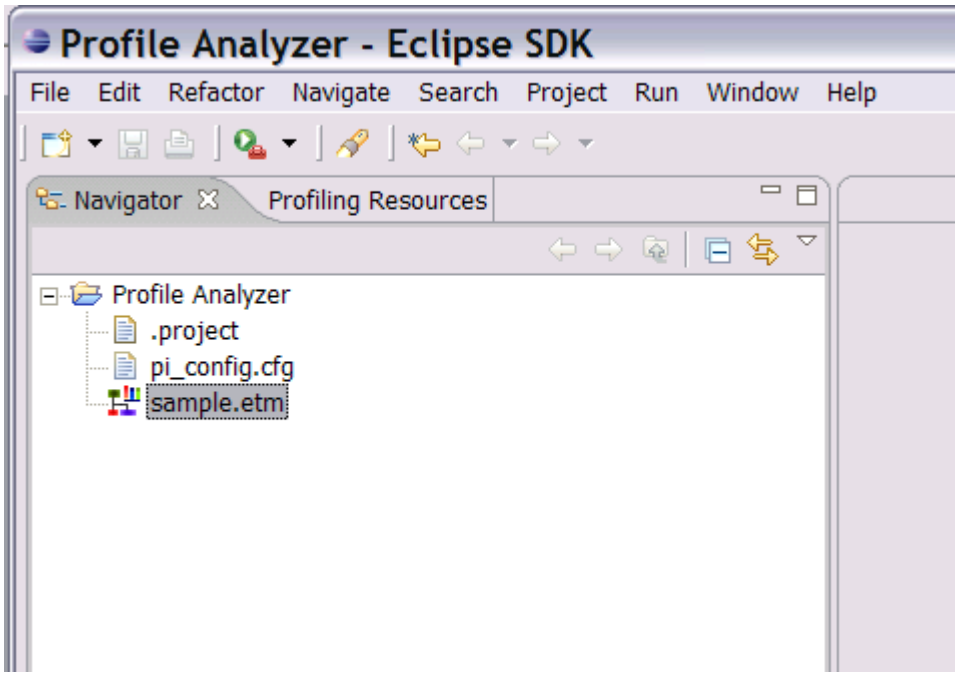


Select Navigator. It will open it as a tab in the bottom right window. Just drag it to the top left where we want it.

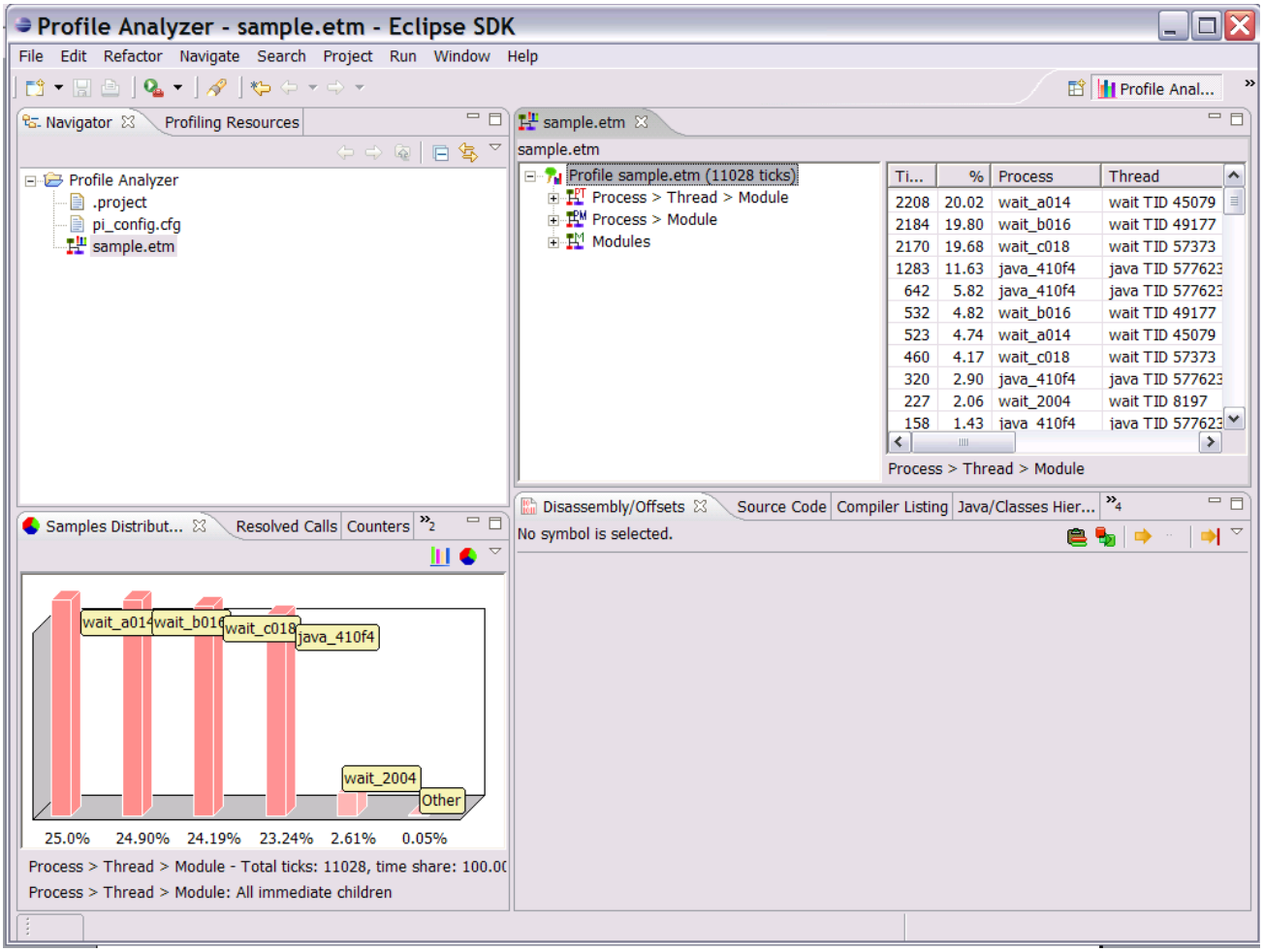


You will then need to navigate to D:\eclipse\workspace\Profile Analyzer to see your data.

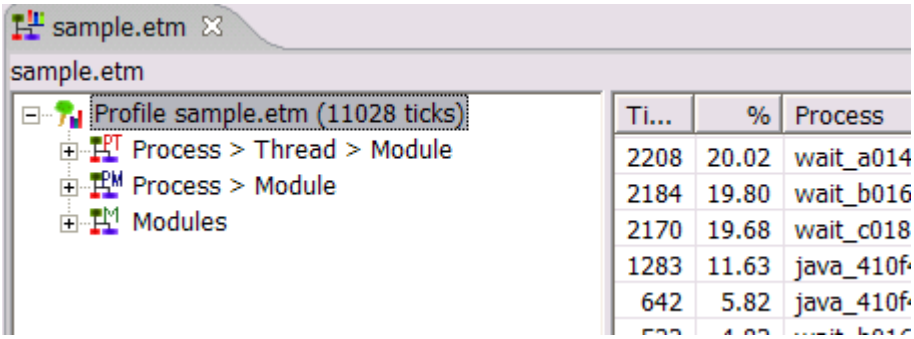
Open the SAMPLE ETM by double clicking on it in the Navigator view.



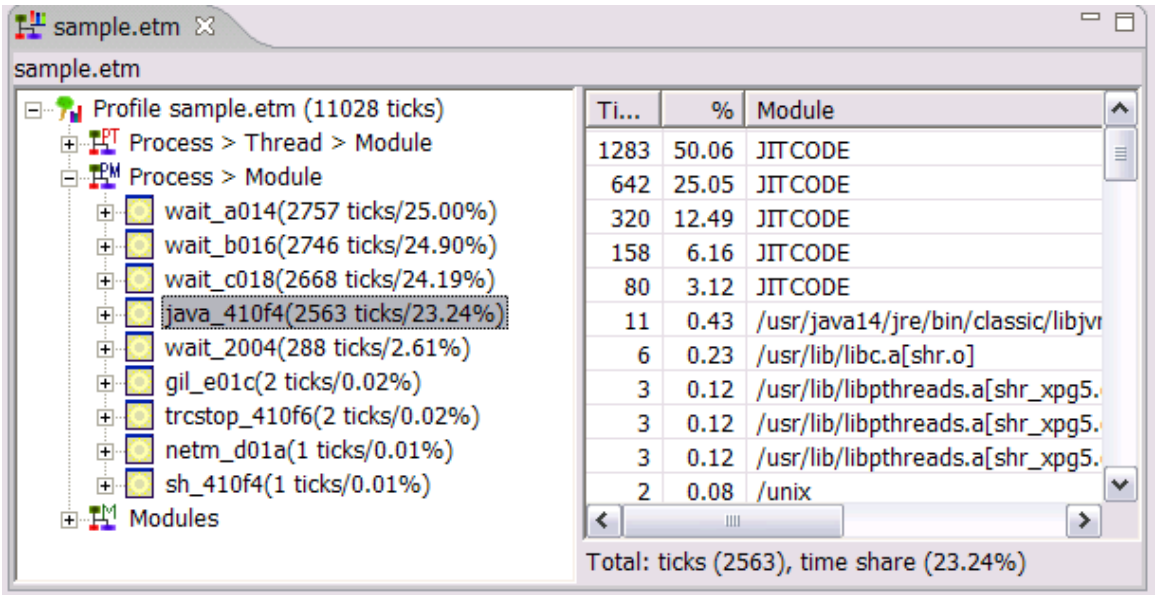
The initial view will look something like this:



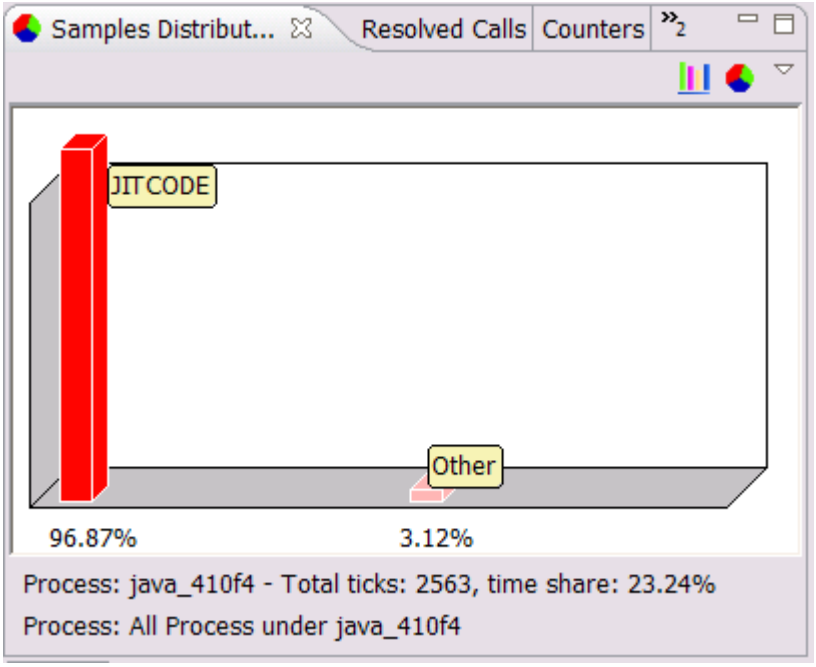
Expand Process > Module on the top right window by clicking the + sign :



Once expanded, we can see that on the 4-way test system, our single threaded Java test case left 3 of the processors idling, leaving the ticks nicely divided amongst the processors. The Java test case (Process ID 410f4) took 23.24% of the total ticks. A ***tick*** is a sampled address recording where the system was executing code. JITCODE is where the work for this Java test case is being done, and we expect this due to Jitting of the Java methods.



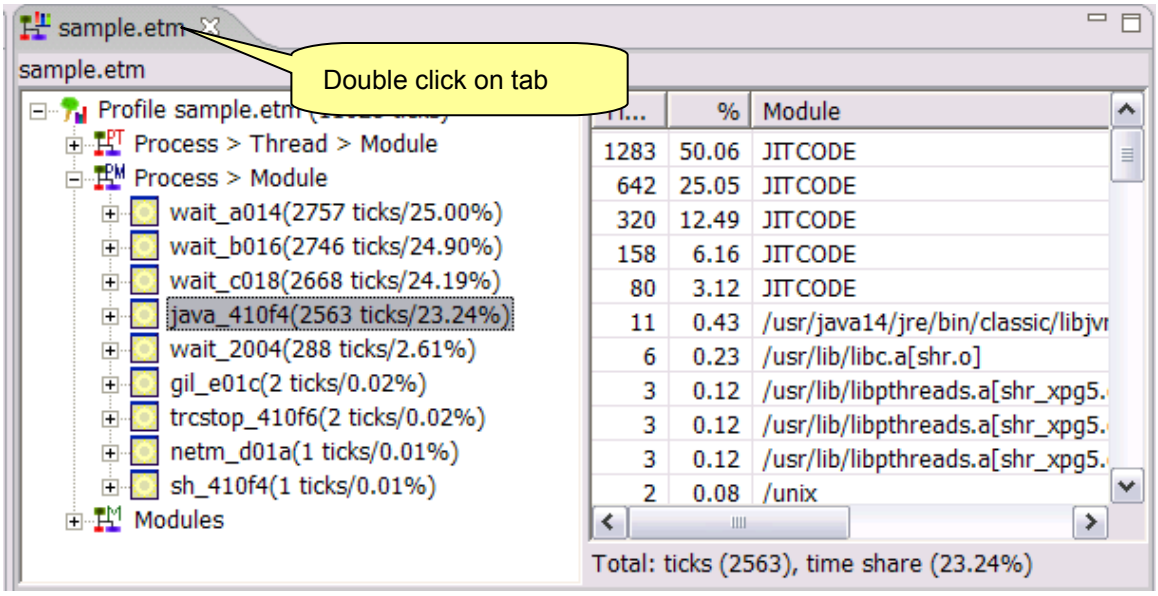
The bar chart on the bottom left of VPA shows this well. Within the Java Process, we took a full 96.87% of the ticks in Jitted code. (If not shown, just click the bar chart icon on the Samples Distribution tab) :



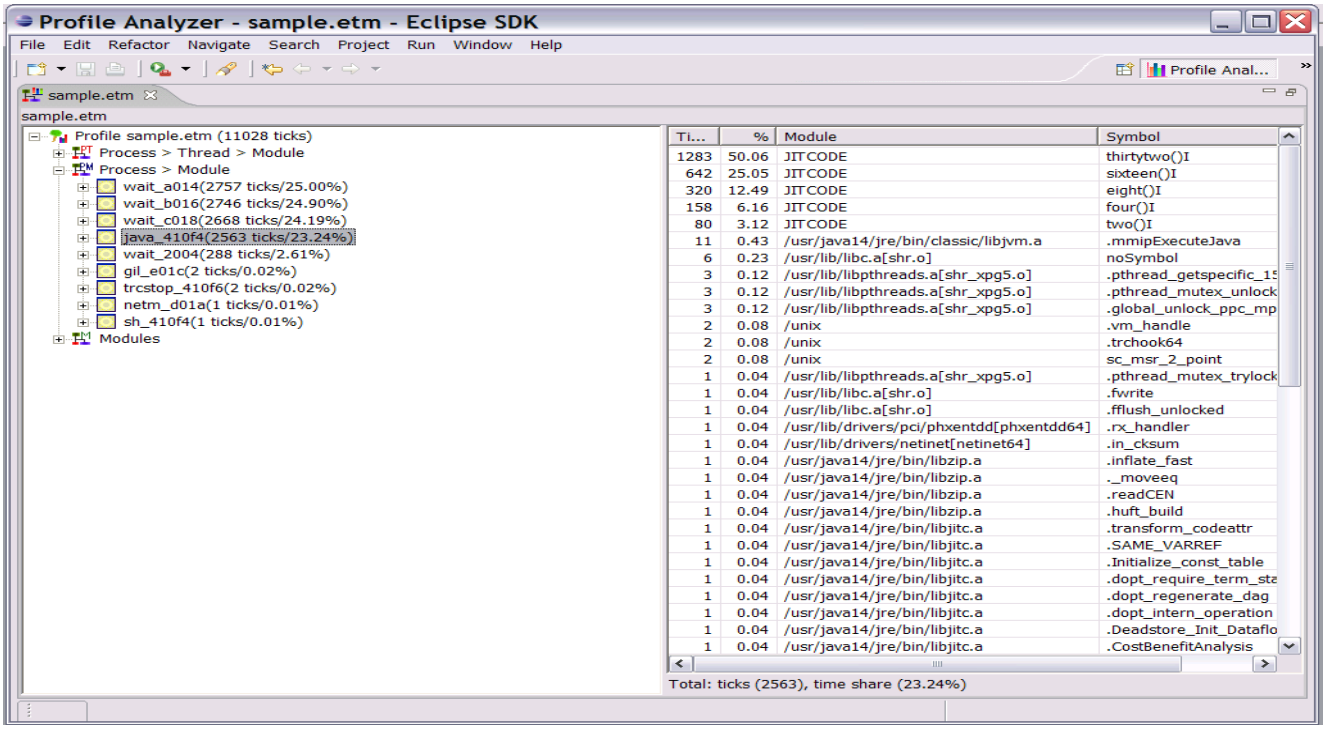
Almost everything with VPA is a mouse click away!!!

Double clicking any tab will change to a maximized view of that tab. Double clicking the same tab will change it back.

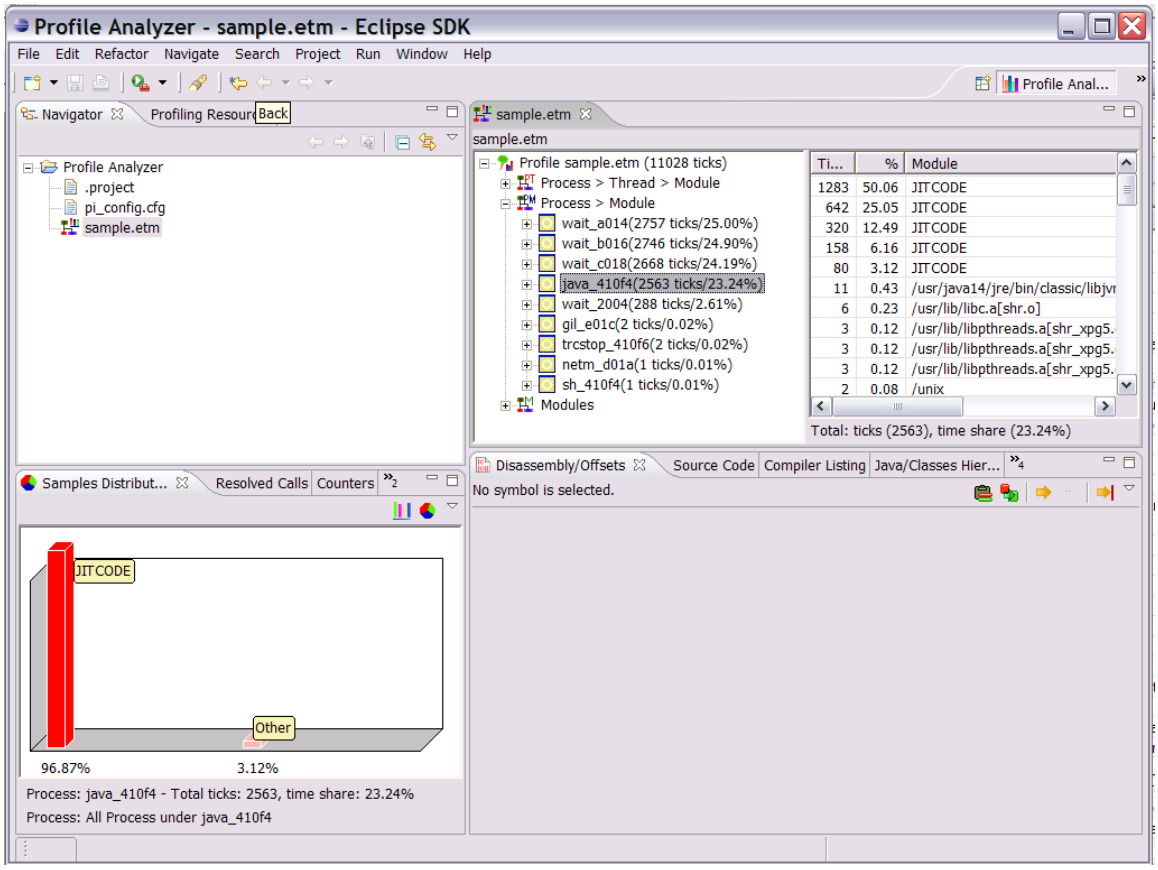
Try it by double clicking the tab for the window on the top right:



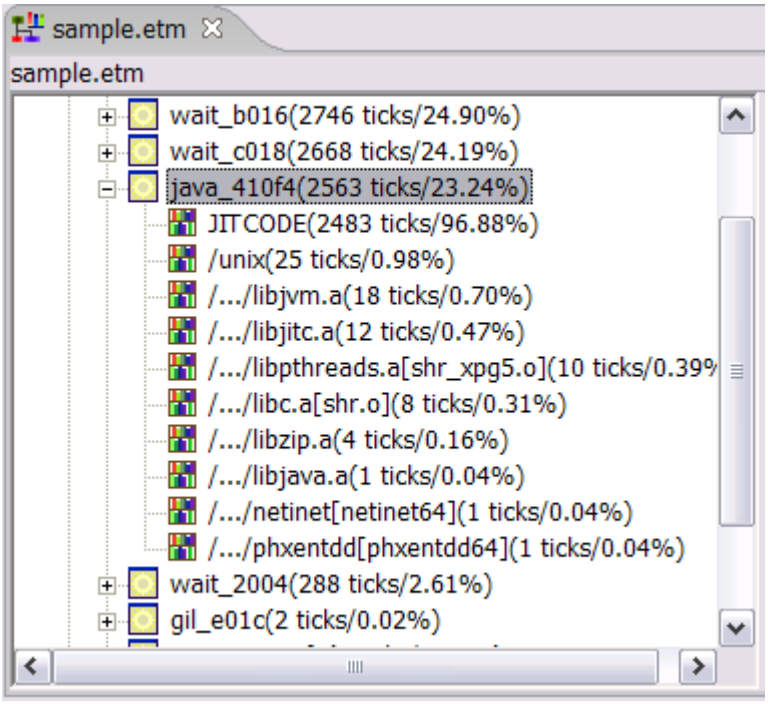
This makes it easier to see everything.



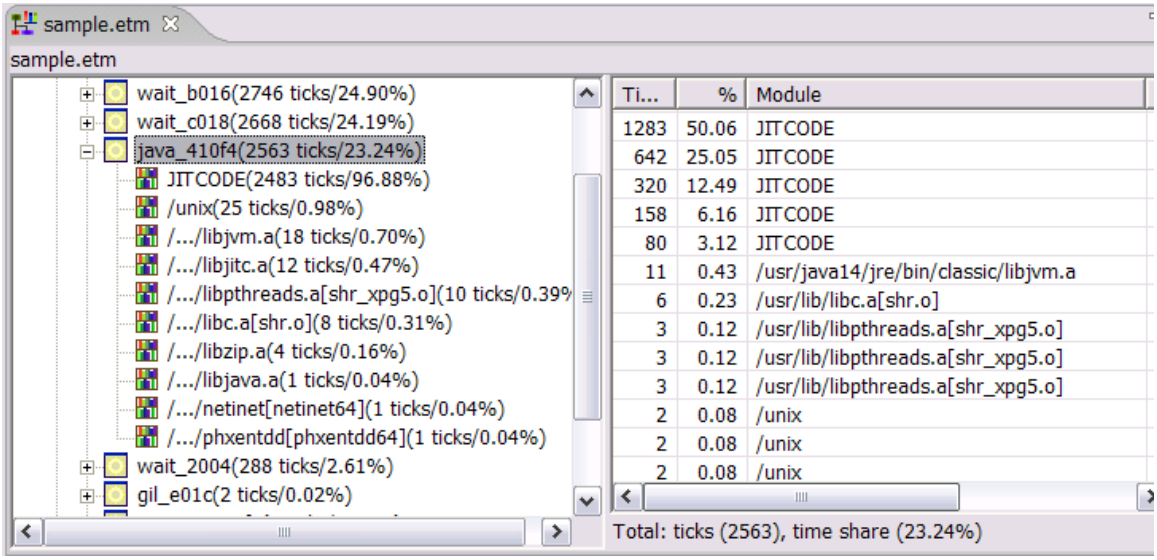
Double click the same tab to go back to the 4 pane view:



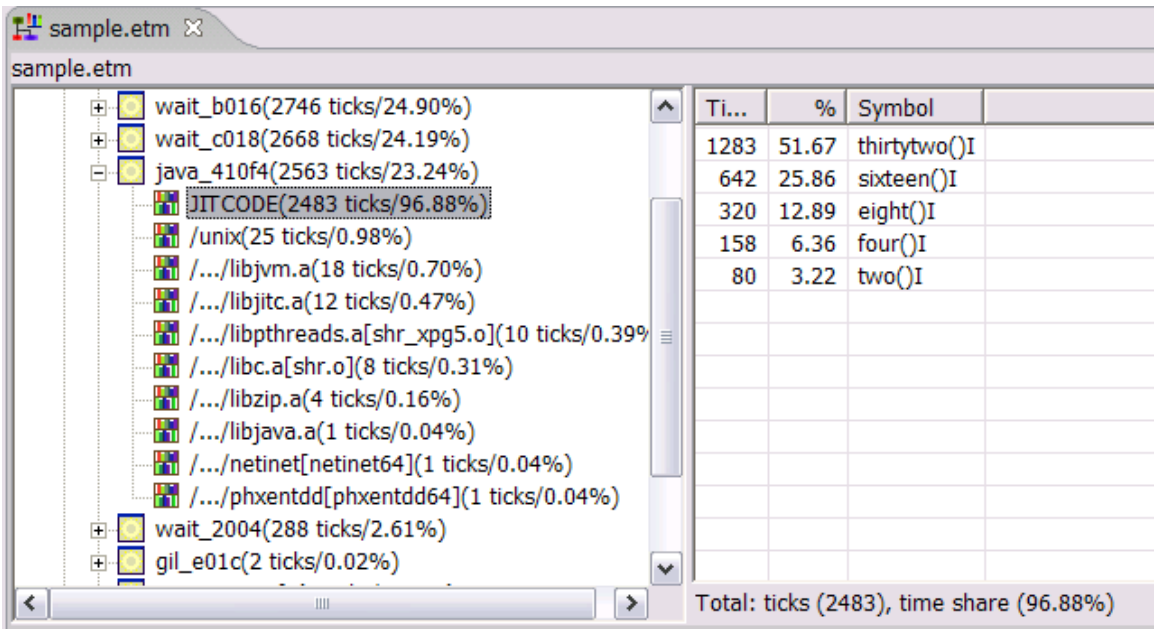
Select PID 410f4 and click the + to expand and see the modules within this process:



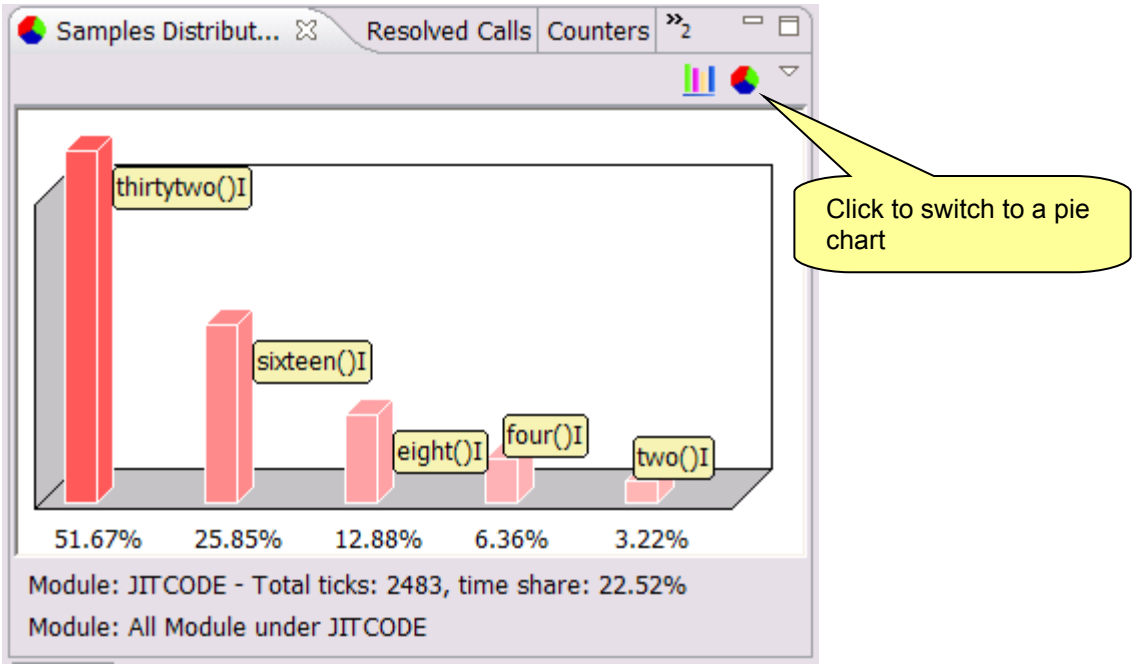
As we noted earlier, the JITCODE module is showing the most ticks as expected for the Java process. On the right hand side we see a breakdown of ticks for the JITCODE module.



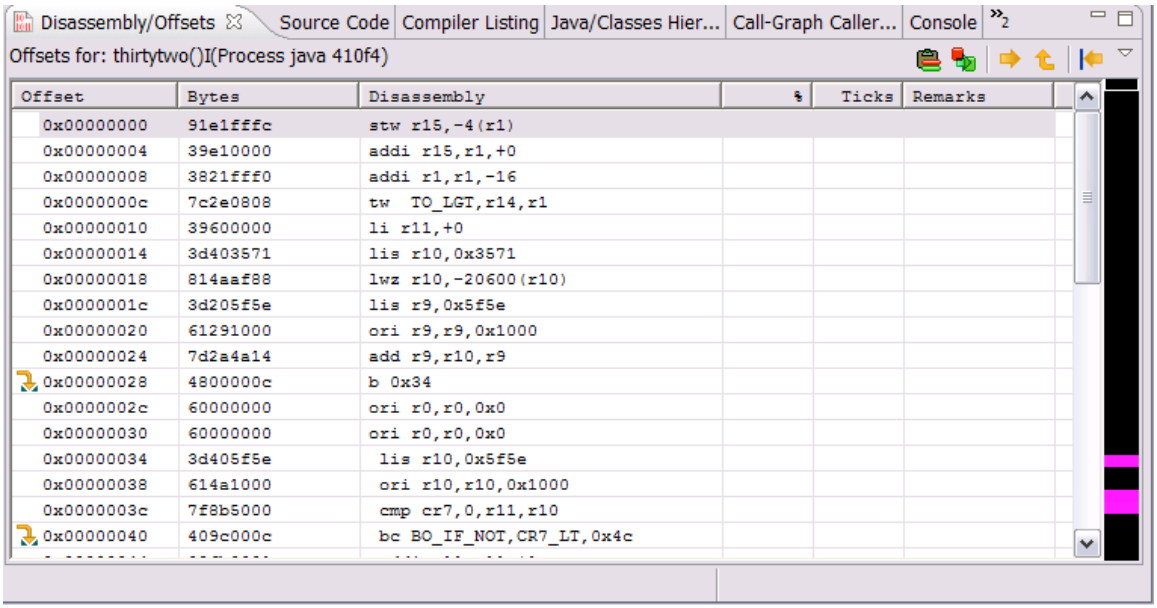
Clicking on the JITCODE Module expands the window on the right, showing a breakdown of ticks in the JITCODE. The hottest symbols are shown in descending order. Notice that thirtytwo() is the biggest contributor at almost 52%:



If you look at the Samples Distribution view in the lower left corner, you will see that the information now corresponds to the JITCODE symbols. You can also switch between pie and bar charts.

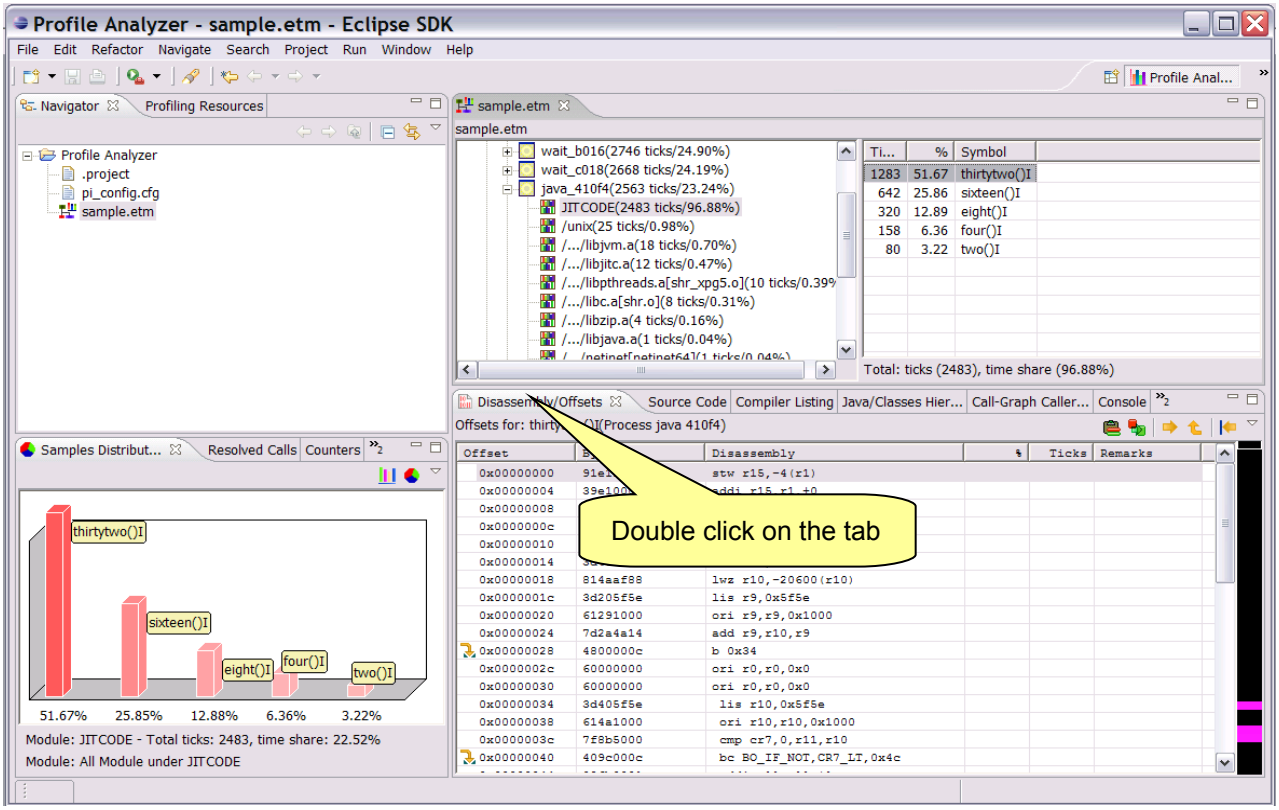


Double click on thirtytwo() and notice that the disassembly/Offset information in the bottom right window is now filled in for this method. Clicking on any other method on the top right will change everything we are seeing in the graph and in this disassembly to correspond to the method selected:

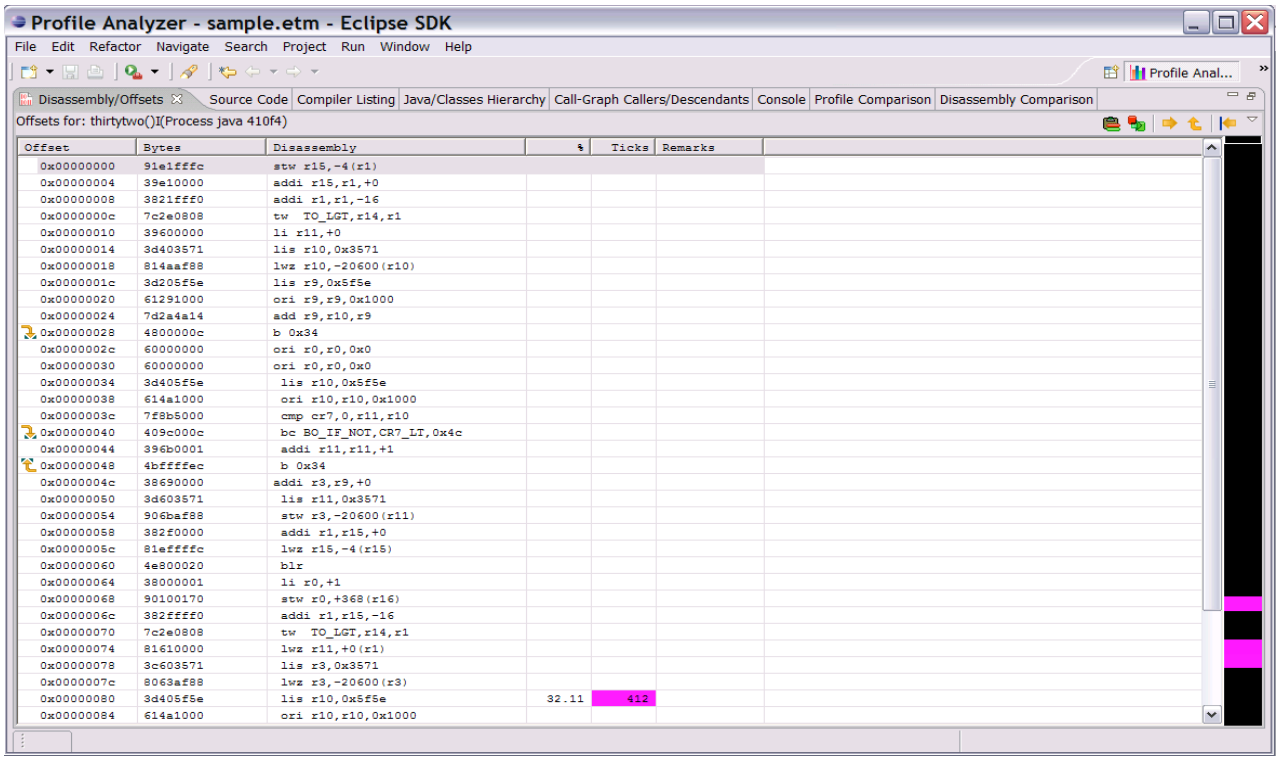


Offset	Bytes	Disassembly	%	Ticks	Remarks
0x00000000	91e1fffc	stw r15,-4(r1)			
0x00000004	39e10000	addi r15,r1,+0			
0x00000008	3821fff0	addi r1,r1,-16			
0x0000000c	7c2e0808	tw TO_LGT,r14,r1			
0x00000010	39600000	li r11,+0			
0x00000014	3d403571	lis r10,0x3571			
0x00000018	814aaf88	lwz r10,-20600(r10)			
0x0000001c	3d205f5e	lis r9,0x5f5e			
0x00000020	61291000	ori r9,r9,0x1000			
0x00000024	7d2a4a14	add r9,r10,r9			
0x00000028	4800000c	b 0x34			
0x0000002c	60000000	ori r0,r0,0x0			
0x00000030	60000000	ori r0,r0,0x0			
0x00000034	3d405f5e	lis r10,0x5f5e			
0x00000038	614a1000	ori r10,r10,0x1000			
0x0000003c	7f8b5000	cmp cr7,0,r11,r10			
0x00000040	409c000c	bc BO_IF_NOT,CR7_LT,0x4c			

We need to get the Disassembly/Offset Information as a full screen, so double click that tab:



We now see a full window view of this:



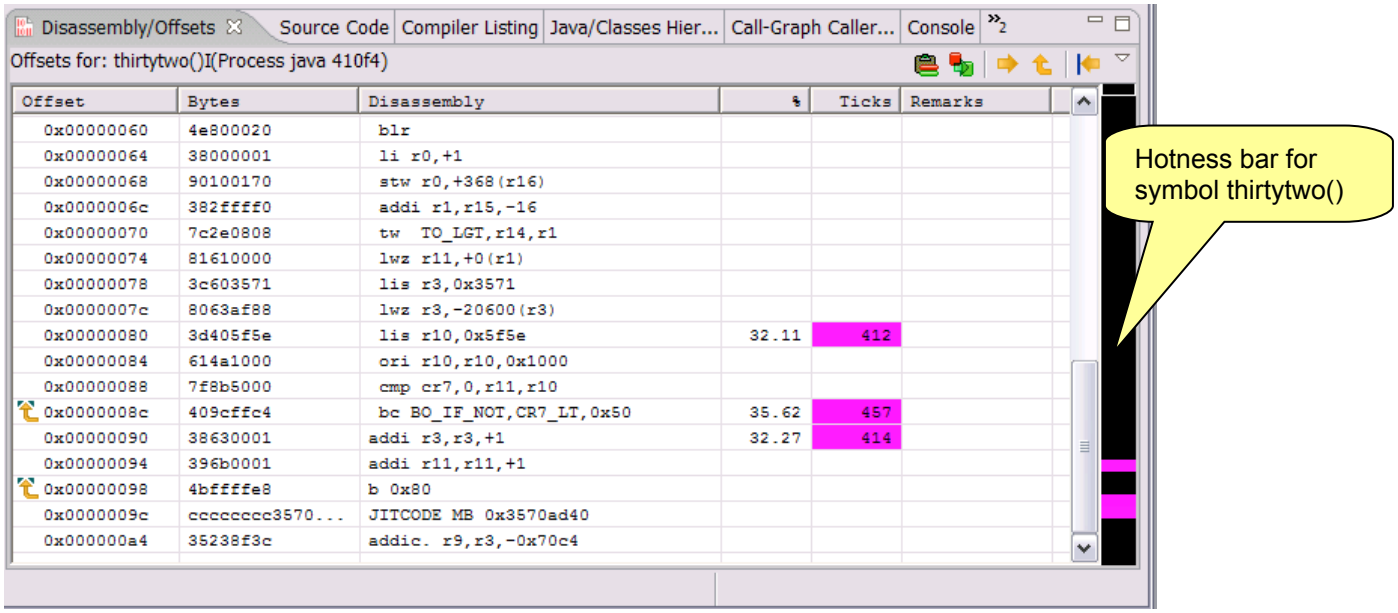
Notice the colored bar on the right. This is the Hotness Bar.

Red is used for any symbol that represents at least 20% of the total tick count.

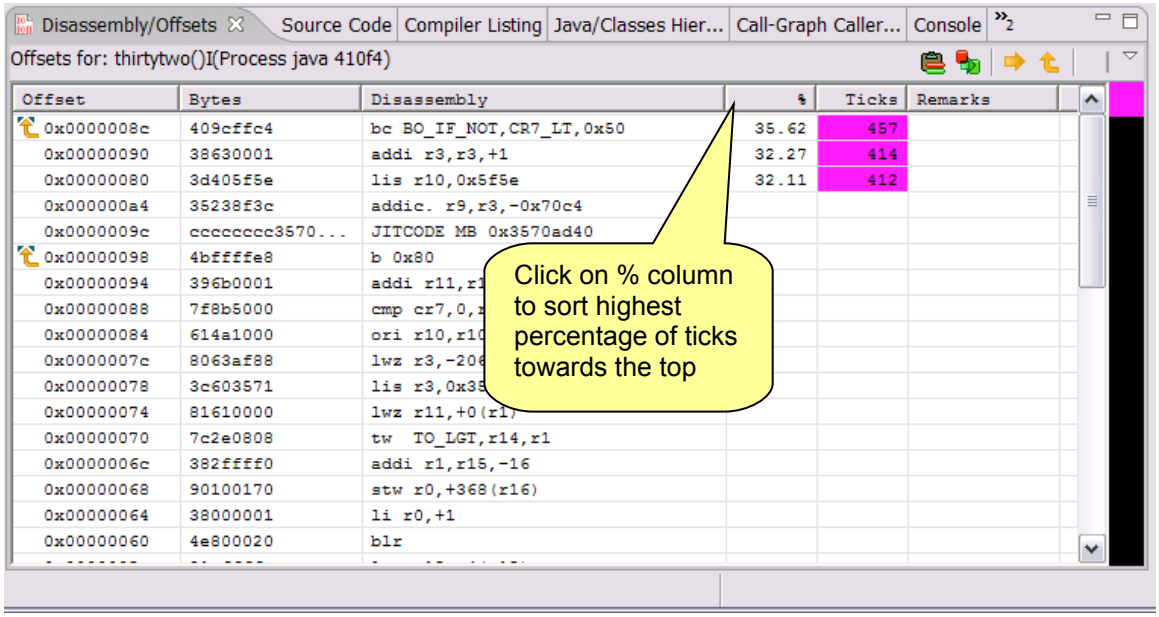
Magenta is used for any symbol that represents between 5% and 20% of the total tick count.

Blue is used for any symbol that uses less than 5% of the total tick count.

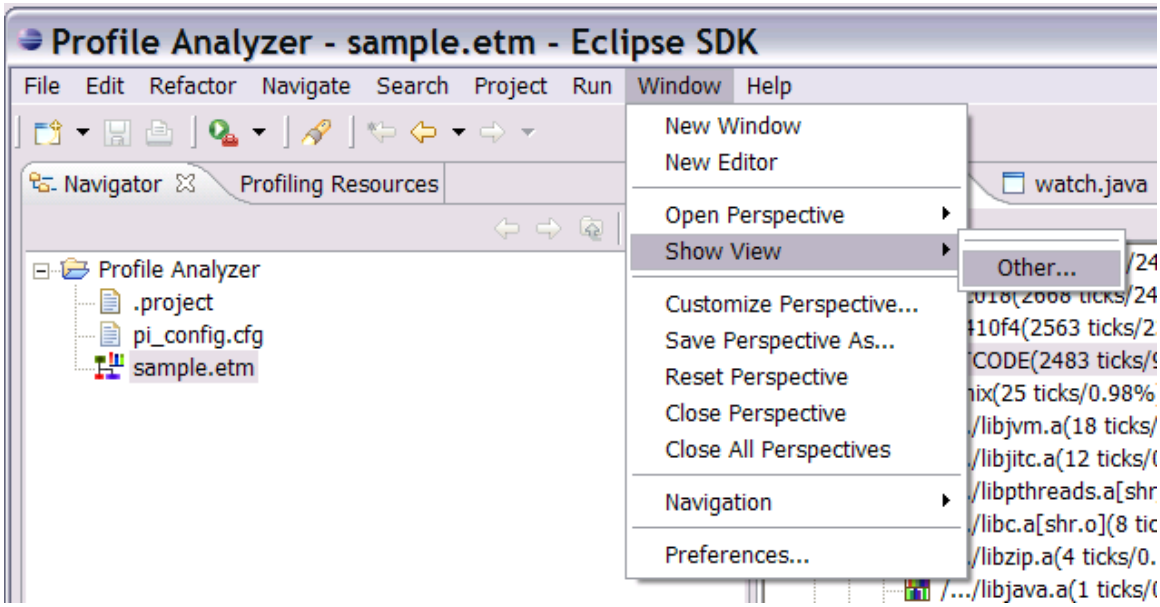
Darker shades show higher activity than lighter shades.



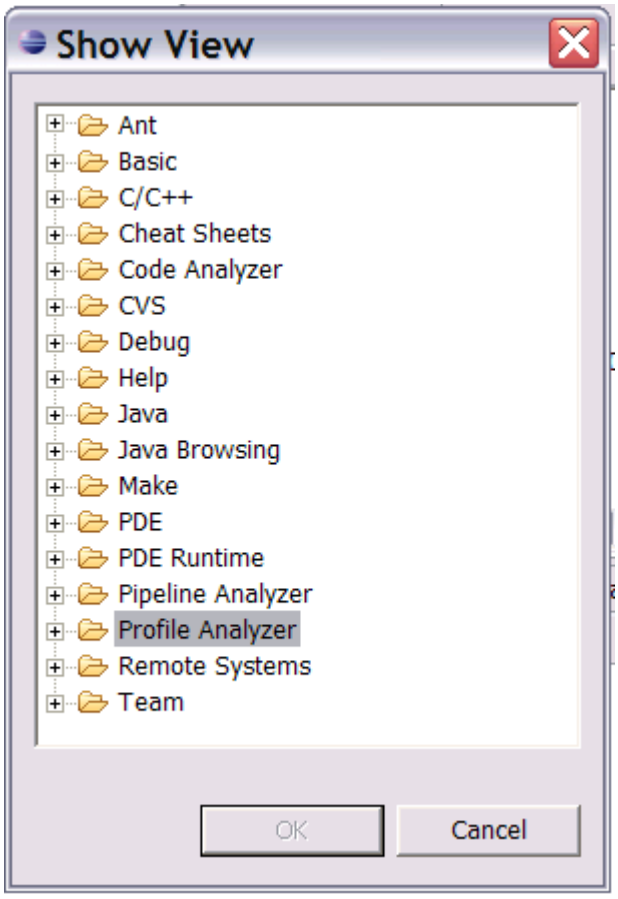
We want to see the code with the most activity, so click on the % column to do a quick sort:



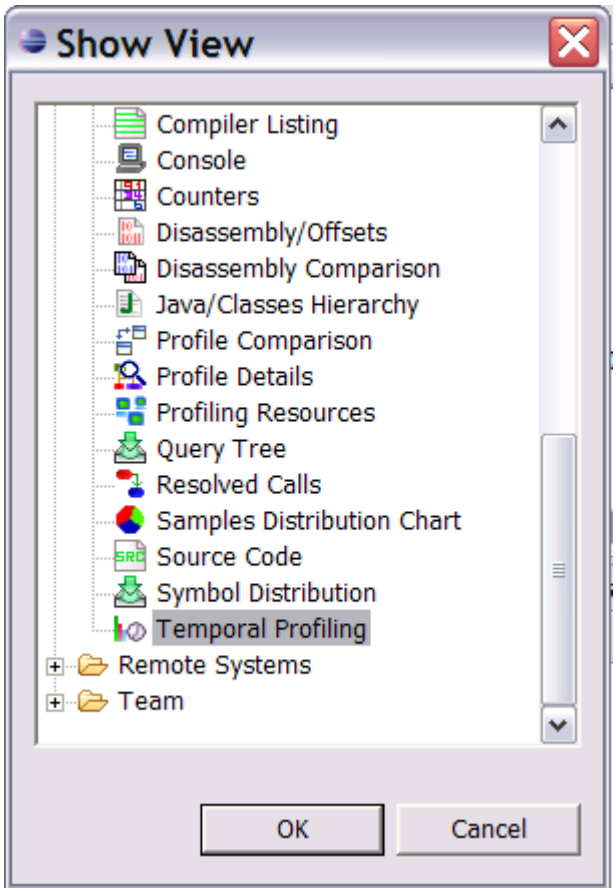
If you would like to use other Profile Analyzer views that is not currently shown. Simply go to Window->Show View -> Other …



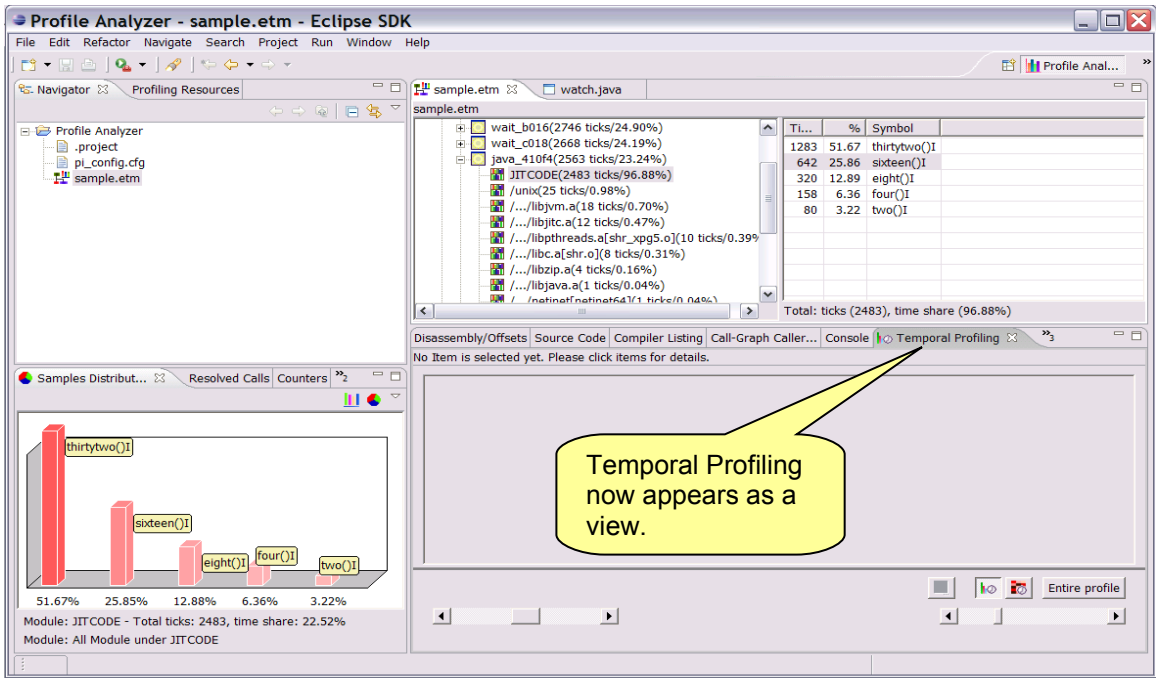
Expand Profile Analyzer to see all the available views …



And as an example .. click on **Temporal Profiling** to load the view…



And you should now have a screen that looks like this …



7.Appendix B – run.tprof_xml.sh

The following script file should be copied to your AIX system. It is required when you create an AIX profile configuration in Profile Analyzer.

```
#!/bin/sh
#
# Remote launch script for AIX tprof with XML generator
#
#
Transfer-encoding: chunked

function CheckForHelp
{
   _RUN_TPROF_HELP=
   case $1 in
      "?" )
         ;&
      "-?" )
         _RUN_TPROF_HELP=short ;;
      "??" )
         ;&
      "-??" )
         ;&
      "/??" )
         _RUN_TPROF_HELP=long ;;
   esac
   if [[ $_RUN_TPROF_HELP != "" ]] ; then
      ShowHelp
   fi
}

function ShowHelp
{
   echo "run.tprof_xml.sh: Execute AIX tprof and generates XML format data file"
   echo
   echo "Syntax:"
   echo "  run.tprof_xml.sh <--start | --stop | --full> <-b #MB> <-s ##> <-r ##> <-R> <-I> <-N> {-m time | -m event -e
## -c ##} <--timedata <--buckets ##>>"
   echo
   echo "Where:"
   echo "------"
   echo "  --start      Start tracing only."
   echo
   echo "  --stop       Stop tracing and generate the XML file."
   echo
   echo "  --full       Run full trace session and generate the XML file"
   echo
   echo "  -b ##        Set trace buffer size to ## MB."
   echo
   echo "  -s ##        Automatically start tracing after ## seconds."
   echo
   echo "  -r ##        Run (trace) for ## seconds."
```

```
    echo
    echo "  -R          Enable PURR weighting."
    echo "              * If you set -R option when using --start, you must also"
    echo "                set it when using --stop."
    echo
    echo "  -I          Turn on binary instructions collecting for tprof2xml."
    echo
    echo "  -N          Turn on source line number info collecting for tprof2xml."
    echo
    echo "  -m {time,event} Select profiling type."
    echo "              * -m time to use timer based profiling."
    echo "              * -m event to use event based profiling."
    echo
    echo "  -E ##       Specify the event to profile."
    echo "              * this option is only valid when -m event is given."
    echo
    echo "  -c ##       Specify the sampling frequency to use."
    echo
    echo "  --timedata   Enable time data generation in XML."
    echo
    echo "  --buckets ## Buckets number for time data. If not specified, default"
    echo "              number is 1800."
    echo "              * --timedata and --buckets options are only valid for"
    echo "                the --full or --stop option."
    echo
    exit 0
}

function DefaultOption
{
    _SCRIPT_NAME=$1
    _TPROF_XML_BUFSIZE=""
    _TPROF_XML_START_DELAY=""
    _TPROF_XML_RUN_TIME=2147483647
    _TPROF_XML_START=""
    _TPROF_XML_STOP=""
    _TPROF_XML_FULL=""
    _TPROF_XML_PURR=""
    _TPROF_XML_TIMEDATA=""
    _TPROF_XML_BUCKETS=""
    _TPROF_XML_INSTR=""
    _TPROF_XML_SRCLINE=""
    _TPROF_XML_TYPE=""
    _TPROF_XML_EVENT=""
    _TPROF_XML_FREQ=""
}


function ParseCmdLine
{
    case $1 in
        "-b" )
            ;&
        "--buffersize" )
            HandleBufferSize $1 $2 ;;
        "-s" )
            HandleS $1 $2 ;;
```

```
      "-r" )
         HandleR $1 $2 ;;
      "--stop" )
         _TPROF_XML_STOP=1 ;;
      "--start" )
         _TPROF_XML_START=1 ;;
      "--full" )
         _TPROF_XML_FULL=1 ;;
      "-R" )
         _TPROF_XML_PURR=1 ;;
      "--timedata" )
         _TPROF_XML_TIMEDATA=1 ;;
      "--buckets" )
         HandleBuckets $1 $2 ;;
      "-I" )
         _TPROF_XML_INSTR=1 ;;
      "-N" )
         _TPROF_XML_SRCLINE=1 ;;
      "-m" )
         HandleType $1 $2 ;;
      "-e" )
         HandleEvent $1 $2 ;;
      "-c" )
         HandleFrequency $1 $2;;
      * )
         echo
         echo "ERROR:" "$1 is not a valid argument. Enter \"run.tprof_xml.sh ? \" for help."
         echo
         exit 1 ;;
   esac
}

function HandleBufferSize
{
   if [[ $2 = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing buffer size with --buffersize option. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_BUFSIZE=$2
}

function HandleS
{
   if [[ $2 = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing time delay with -s option. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_START_DELAY=$2
}

function HandleR
{
   if [[ $2 = "" ]] ; then
```

```
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing run time with -r option. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_RUN_TIME=$2
}

function HandleBuckets
{
   if [[ $2 = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing buckets number. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_BUCKETS=$2
}

function HandleType
{
   if [[ $2 = "time" || $2 = "event" ]] ; then
      _TPROF_XML_TYPE=$2
   fi
   if [[ $_TPROF_XML_TYPE = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Invalid profiling type. Only -m time and -m event are supported. Quitting."
      echo
      exit 1
   fi
}

function HandleEvent
{
   if [[ $2 = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing event with -e option. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_EVENT=$2
}

function HandleFrequency
{
   if [[ $2 = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": Missing event frequency with -c option. Quitting."
      echo
      exit 1
   fi
   _TPROF_XML_FREQ=$2
}

function PostProcessing
{
   if [[ ! -f rse_trace.trc || ! -f rse_trace.syms ]] ; then
```

```
        echo
        echo "ERROR:" $_SCRIPT_NAME": tprof trace files not found. Quitting"
        echo
        exit 1
    fi

    #
    # call the XML generator
    #
    OPTIONS=""
    if [[ $_TPROF_XML_PURR != "" ]] ; then
        OPTIONS=$OPTIONS" -R "
    fi
    if [[ $_TPROF_XML_TIMEDATA != "" ]] ; then
        OPTIONS=$OPTIONS" --timedata "
    fi
    if [[ $_TPROF_XML_BUCKETS != "" ]] ; then
        OPTIONS=$OPTIONS" --buckets "$_TPROF_XML_BUCKETS
    fi

    /usr/lib/perf/tprof2xml $OPTIONS -c rse_trace.trc -s rse_trace.syms -o out.etm

    echo '##_done_get_the_file##'
}

function Cleanup
{
    rm -f rse_trace.*
    unset _SCRIPT_NAME
    unset _TPROF_XML_BUFSIZE
    unset _TPROF_XML_START_DELAY
    unset _TPROF_XML_RUN_TIME
    unset _TPROF_XML_STOP
    unset _TPROF_XML_START
    unset _TPROF_XML_FULL
    unset _TPROF_XML_PURR
    unset _TPROF_XML_TIMEDATA
    unset _TPROF_XML_BUCKETS
    unset _TPROF_XML_INSTR
    unset _TPROF_XML_SRCLINE
    unset _TPROF_XML_TYPE
    unset _TPROF_XML_EVENT
    unset _TPROF_XML_FREQ
}

#
# Script Entry ...
#
# (1) parse for cmd line options
# (2) validate options
# (3) invoke proper sub command: START | STOP | FULL
#
# START & FULL ... kick off Tprof in background
#
# STOP ... waits for background Tprof to exit and then runs tprof2xml ... output is OUT.ETM file
#
```

```
CheckForHelp $1
DefaultOption $0
while [[ $1 != "" ]]
do
   ParseCmdLine $1 $2
   if [[ $1 = "-b" || $1 = "-s" || $1 = "-r" || $1 = "--buckets" || $1 = "-m" || $1 = "-e" || $1 = "-c" ]] ; then
      shift
   fi
   shift
done

###
### Validate Options
###
if [[ $_TPROF_XML_START = "" && $_TPROF_XML_STOP = "" && $_TPROF_XML_FULL = "" ]] ; then
   ShowHelp
   exit 1
fi

if [[ $_TPROF_XML_TYPE = "" ]] ; then
   echo
   echo "ERROR:" $_SCRIPT_NAME": Profiling type must be specified by -m time or -m event. Quitting."
   echo
   exit 1
fi

if [[ $_TPROF_XML_TYPE = "time" ]] ; then
   if [[ $_TPROF_XML_EVENT != "" || $_TPROF_XML_FREQ != "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": -m time can not work with -e or -c options. Quitting."
      echo
      exit 1
   fi
fi

if [[ $_TPROF_XML_TYPE = "event" ]] ; then
   if [[ $_TPROF_XML_EVENT = "" || $_TPROF_XML_FREQ = "" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": -m event must be given together with -e and -c options. Quitting."
      echo
      exit 1
   fi
fi


if [[ $_TPROF_XML_EVENT != "" || $_TPROF_XML_FREQ != "" ]] ; then
   if [[ $_TPROF_XML_TYPE != "event" ]] ; then
      echo
      echo "ERROR:" $_SCRIPT_NAME": -e or -c must be given together with -m event. Quitting."
      echo
      exit 1
   fi
fi

###
### STOP Command
###
```

```
if [[ $_TPROF_XML_STOP != "" ]] ; then
   _SLEEP_PID=`ps -e | awk '$NF == "sleep" { print $1; exit }'`
   if [[ $_SLEEP_PID = "" ]] ; then
      echo "ERROR:" "PID not found."
      exit 1
   fi
   kill $_SLEEP_PID

   ###
   ### wait for tprof to finish
   ###
   _TPROF_PID=`ps -e | awk '$NF == "tprof" { print $1; exit }'`
   while [[ $_TPROF_PID != "" ]]
   do
      sleep 2
      _TPROF_PID=`ps -e | awk '$NF == "tprof" { print $1; exit }'`
   done

   ###
   ### do post process
   ###
   PostProcessing
   Cleanup

   echo
   echo Done
   echo

   echo '##_done_stop_tools##'

   exit 0
fi


#
#  START or FULL command
#
if [[ $_TPROF_XML_START != "" || $_TPROF_XML_FULL != "" ]] ; then
   ###
   ### kill previous not finished tprof process
   ###
   _TPROF_PID=`ps -e | awk '$NF == "tprof" { print $1; exit }'`
   while [[ $_TPROF_PID != "" ]]
   do
      /usr/bin/trcstop
      kill $_TPROF_PID
      sleep 1
      _TPROF_PID=`ps -e | awk '$NF == "tprof" { print $1; exit }'`
   done


   ###
   ### kill the sleep process launched by tprof
   ###
   _SLEEP_PID=`ps -e | awk '$NF == "sleep" { print $1; exit }'`
   while [[ $_SLEEP_PID != "" ]]
   do
```

```
      kill $_SLEEP_PID
      sleep 1
      _SLEEP_PID=`ps -e | awk '$NF == "sleep" { print $1; exit }'`
   done
fi

###
### start delay
###
if [[ $_TPROF_XML_START_DELAY != "" ]] ; then
   sleep $_TPROF_XML_START_DELAY
fi

###
### Launch tprof
###
rm -f rse_trace.*
rm -f out.etm
OPTIONS=""
if [[ $_TPROF_XML_PURR != "" ]]; then
   OPTIONS=$OPTIONS" -R "
fi
if [[ $_TPROF_XML_INSTR != "" ]]; then
   OPTIONS=$OPTIONS" -I "
fi
if [[ $_TPROF_XML_SRCLINE != "" ]]; then
   OPTIONS=$OPTIONS" -N "
fi
if [[ $_TPROF_XML_TYPE = "event" ]] ; then
   OPTIONS=$OPTIONS" -E "$_TPROF_XML_EVENT" -f "$_TPROF_XML_FREQ" "
fi

if [[ $_TPROF_XML_START != "" ]]; then
   tprof -eukj -A -F $OPTIONS -r rse_trace -x sleep $_TPROF_XML_RUN_TIME &
   echo '##_done_start_tools##'
   exit 0
else
   echo '##_done_start_tools##'
   tprof -eukj -X -A -F $OPTIONS -r rse_trace -x sleep $_TPROF_XML_RUN_TIME
fi

PostProcessing
Cleanup

echo
echo Done
echo

echo '##_done_start_tools##'
exit 0
```

8.Appendix C – run.tprof_e.cmd

The following cmd file should be copied to your Performance Inspector for Windows bin directory. It is required when you create Windows profile configuration in Profile Analyzer.

```
@setlocal
@echo off
   set _TPROF_POST_RC=0

   echo.
   echo ***** run.tprof v2.06 *****
   echo.
   goto CheckForHelp

:Help
   echo run.tprof: Coordinates the TPROF procedure
   echo.
   echo Syntax:
   echo -------
   echo    run.tprof ^<-a ^| -t ^| -p^> ^<-b #MB^> ^<-m [time ^| event]^> ^<-s ##^> ^<-r ##^>
   echo              Options valid with '-m time':
   echo                ^<-c #ticks_per_second^>
   echo              Options valid with '-m event':
   echo                ^<-e event_name^> ^<-c #events^>
   echo.
   if "%_RUN_TPROF_HELP%" == "long" goto ContinueHelp
   echo    Enter "run.tprof ??" for more extensive help.
   echo.
   goto Exit

:ContinueHelp
   echo Where:
   echo ------
   echo   -a       Run entire TPROF procedure.
   echo            * !!! THIS IS THE DEFAULT AND NEED NOT BE SPECIFIED !!!
   echo            * Performs the trace and post-processing steps.
   echo            * Produces a raw trace file (swtrace.nrm2) and a TPROF
   echo              report file (tprof.out).
   echo            * This option is mutually exclusive with -t and -p.
   echo.
   echo   -t       Run tracing only.
   echo            * Performs only the tracing (ie. data collection)
   echo              step of the TPROF procedure.
   echo            * Produces a raw trace file (swtrace.nrm2).
   echo            * The trace will not be post-processed. You can
   echo              post-process the trace at a later time by invoking
   echo              run.tprof with the -p option.
   echo            * This option is mutually exclusive with -a and -p.
   echo.
   echo   -p       Run post-processing only.
   echo            * Performs only the post-processing step of the TPROF
```

```
echo            procedure.
echo          * It requires a trace file (swtrace.nrm2) and assumes
echo            that you've already done the tracing step, either by
echo            itself (via a previous -t) or in a previous
echo            complete run (via -a).
echo          * Produces a TPROF report file (tprof.out).
echo          * This option is mutually exclusive with -a and -t.
echo.
echo  -b ##     Set trace buffer size to ## MB **PER PROCESSOR**
echo          * You can also set this value using the
echo            IBMPERF_TPROF_BUFFER_SIZE environment variable
echo            in the toolsenv.cmd configuration file.
echo.
echo  -m mode   Set the TPROF mode.
echo          * Valid modes are:
echo            - TIME
echo              This causes TPROF to run in TIME-based mode.
echo              In this mode sampling is based on time and the time
echo              between samples is constant. The sampling rate
echo              can set via the -c option.
echo            - EVENT
echo              This causes TPROF to run in EVENT-based mode.
echo              In this mode sampling is based on event counts and
echo              the time between samples is not constant. The sampling
echo              event count can set via the -c option.
echo          * Default mode is time.
echo          * You can also set this value using the
echo            IBMPERF_TPROF_MODE environment variable in the
echo            toolsenv.cmd configuration file.
echo.
echo  -e event  Set event name when in EVENT-based mode.
echo          * 'event' is the name of an event. Valid event names
echo            can be obtained by entring the "SWTRACE EVENT LIST"
echo            command.
echo            - Event names are not case sensitive. INSTR is the same
echo              as Instr or iNStr or instr or ...
echo            - An event count can be set using the -c option.
echo          * Default event name is INSTR.
echo          * You can also set this value using the
echo            IBMPERF_TPROF_EVENT environment variable in the
echo            toolsenv.cmd configuration file.
echo.
echo  -c ##     Set tick rate (TIME-based) or event count (EVENT-based).
echo          * If mode is TIME then ## represents the desired TPROF tick
echo            rate, in ticks/second.
echo            - Default tick rate is 128 ticks/second.
echo            - You can also set this value using the
echo              IBMPERF_TPROF_TICK_RATE environment variable in the
echo              toolsenv.cmd configuration file.
echo          * If mode is EVENT then ## represent the number of events
echo            after which a TPROF tick will occur.
echo            - Default event count is 10,000,000 events.
echo            - You can also set this value using the
echo              IBMPERF_TPROF_EVENTCNT environment variable in the
echo              toolsenv.cmd configuration file.
echo          * Setting the rate too high may affect overall system
echo            performance.
```

```
echo          * Setting the rate too low may not collect enough samples.
echo.
echo   -s ##    Automatically start tracing after ## seconds.
echo          * You WILL NOT be prompted for when to start tracing.
echo            Instead, tracing will start automatically after a
echo            delay of ## seconds.
echo          * If you DO NOT specify the -r option, you will be
echo            prompted for when to stop tracing.
echo          * If you specify 0 (zero), or a non-numeric value,
echo            tracing is started immediately.
echo          * If you use this option your application should be
echo            started and warmed up by the time tracing is started.
echo          * Only applicable with -a or -t. Ignored otherwise.
echo          * Default is to prompt for when to start tracing.
echo.
echo   -r ##    Run (trace) for ## seconds.
echo          * You WILL NOT be prompted for when to stop tracing.
echo            Instead, tracing will stopt automatically after a
echo            delay of ## seconds from the time tracing was started.
echo          * If you DO NOT specify the -s option, tracing is stopped
echo            ## seconds after you press ENTER to start the trace.
echo          * If you specify 0 (zero), or a non-numeric value,
echo            tracing is stopped immediately after being started.
echo          * If you use this option your application should
echo            complete the scenario you want to trace in the
echo            amount of time you specify with this option.
echo          * Only applicable with -a or -t. Ignored otherwise.
echo          * Default is to prompt for when to stop tracing.
echo.
echo Notes:
echo ------
echo   * Options are parsed from left to right. If you specify an
echo     option more than once, the rightmost option is used.
echo   * -a, -t and -p are exclusive options:
echo     only one of them is allowed.
echo   * -b, -m, -c and -e override the values set in 'toolsenv.cmd'
echo     environment variables.
echo   * The meaning of -c varies depending on the mode. It is assumed
echo     to be a rate (ticks/second) in TIME-based mode, or an event
echo     count in EVENT-based mode.
echo   * -e is ignored if the current mode is TIME.
echo   * -s and -r are useful if you are automating the TPROF
echo     procedure and have an idea of how long it takes for the
echo     scenario to start up (-s option) and how long you want
echo     to TPROF for (-r option).
echo   * Valid option combinations:
echo     No options      Run entire procedure. Prompt to start
echo                   and stop trace.
echo     -a             Same as no options.
echo                   Trace start/stop controlled by -s/-r.
echo     -t             Trace only. No post-processing.
echo                   Trace start/stop controlled by -s/-r.
echo     -p             Post-process only. No tracing.
echo.
echo     No -s and no -r  Prompt to start and prompt stop tracing.
echo     -s ##          Automatically start tracing in ## seconds.
echo                   Prompt to stop.
```

```
echo    -r ##           Prompt to start tracing. Automatically stop
echo                    tracing ## seconds after tracing started.
echo    -s ## -r ##     Automatically start tracing in ## seconds.
echo                    Automatically stop tracing ## seconds after
echo                    tracing started.
echo  * Windows batch language does not provide a way to check
echo    whether a value is a valid number or not. As such, the
echo    values entered with the -s, -r, -b and -c options are not
echo    checked. A non-numeric value is the same as entering 0 (zero).
echo  * run.tprof will not run if environment variable
echo    IBMPERF_TOOLS_PATH isn't set.
echo  * All customization should be done via TOOLSENV.CMD. You should
echo    not need to modify this command file.
echo.
echo Examples:
echo ---------
echo  1) run.tprof
echo     Runs the entire TPROF procedure. You are prompted to start
echo     and stop the trace. Generates TPROF report.
echo     Same as:  'run.tprof -a'
echo.
echo  2) run.tprof -s 10 -r 60
echo     Runs the entire TPROF procedure. Tracing starts automatically
echo     after a 10 seconds delay. Tracing stops automatically 60
echo     seconds after being started. Generates TPROF report. There
echo     are no prompts - everything happens automatically.
echo     Same as: 'run.tprof -a -s 10 -r 60'
echo.
echo  3) run.tprof -t
echo     Runs the trace step of the TPROF procedure. You are
echo     prompted to start and stop the trace. Does not generate a
echo     TPROF report.
echo.
echo  4) run.tprof -t -s 10 -r 60
echo     Runs the trace step of the TPROF procedure. Tracing starts
echo     automatically after a 10 seconds delay. Tracing stops
echo     automatically 60 seconds after being started. There are no
echo     prompts - everything happens automatically.
echo.
echo  5) run.tprof -t -s 10
echo     Runs the trace step of the TPROF procedure. Tracing starts
echo     automatically after a 10 seconds delay. You are prompted for
echo     when to stop tracing.
echo.
echo  6) run.tprof -p
echo     Runs the post-processing step of the TPROF procedure.
echo     You are prompted to start and stop the trace. Only generates
echo     a TPROF report.
echo.
echo  7) run.tprof -m event -e l2_read_refs -c 100000 s 10 -r 60
echo     Runs the entire TPROF procedure. Sampling is EVENT-based, using
echo     event L2_READ_REFS and sampling every 100,000 occurences.
echo     Tracing starts automatically after a 10 seconds delay. Tracing
echo     stops automatically 60 seconds after being started. Generates
echo     TPROF report. There are no prompts - everything happens
echo     automatically.
echo.
```

```
   goto Exit

Rem ************************************************************************
Rem ************************************************************************
Rem                      Command line parsing
Rem ************************************************************************
Rem ************************************************************************

:CheckForHelp
Rem **
Rem ** See if they want help
Rem **
   SET _RUN_TPROF_HELP=
   if "%1" == "?"   SET _RUN_TPROF_HELP=short
   if "%1" == "-?"  SET _RUN_TPROF_HELP=short
   if "%1" == "/?"  SET _RUN_TPROF_HELP=short
   if "%1" == "??"  SET _RUN_TPROF_HELP=long
   if "%1" == "-??" SET _RUN_TPROF_HELP=long
   if "%1" == "/??" SET _RUN_TPROF_HELP=long
   if not "%_RUN_TPROF_HELP%" == "" goto Help

Rem **
Rem ** Set defaults for command line options
Rem **
   SET _TPROF_START_DELAY=
   SET _TPROF_RUN_TIME=
   SET _TPROF_RUN_TYPE=
   SET _TPROF_BUFSIZE=
   SET _TPROF_MODE=
   SET _TPROF_EVENT=
   SET _TPROF_CNT=

Rem **
Rem ** Parse command line
Rem **
:ParseCmdLine
    if "%1" == ""   goto DoneParsing
    if "%1" == "-a" goto HandleAll
    if "%1" == "-A" goto HandleAll
    if "%1" == "-t" goto HandleTrace
    if "%1" == "-T" goto HandleTrace
    if "%1" == "-p" goto HandlePost
    if "%1" == "-P" goto HandlePost
    if "%1" == "-s" goto HandleS
    if "%1" == "-S" goto HandleS
    if "%1" == "-r" goto HandleR
    if "%1" == "-R" goto HandleR
    if "%1" == "-b" goto HandleB
    if "%1" == "-B" goto HandleB
    if "%1" == "-m" goto HandleM
    if "%1" == "-M" goto HandleM
    if "%1" == "-e" goto HandleE
    if "%1" == "-E" goto HandleE
    if "%1" == "-c" goto HandleC
    if "%1" == "-C" goto HandleC
    echo.
    echo "%1" is not a valid argument. Enter "run.tprof ?" for help.
```

```
    goto Exit
:ParseContinue
    shift
    goto ParseCmdLine

Rem **
Rem ** Handle -a optiom
Rem **
:HandleAll
    if not "%_TPROF_RUN_TYPE%" == "" goto All999
    SET _TPROF_RUN_TYPE=-a
    goto ParseContinue
:All999
    if "%_TPROF_RUN_TYPE%" == "-a" goto ParseContinue
    echo.
    echo run.tprof: %1 not allowed with %_TPROF_RUN_TYPE% option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -t optiom
Rem **
:HandleTrace
    if not "%_TPROF_RUN_TYPE%" == "" goto Trace999
    SET _TPROF_RUN_TYPE=-t
    goto ParseContinue
:Trace999
    if "%_TPROF_RUN_TYPE%" == "-t" goto ParseContinue
    echo.
    echo run.tprof: %1 not allowed with %_TPROF_RUN_TYPE% option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -p optiom
Rem **
:HandlePost
    if not "%_TPROF_RUN_TYPE%" == "" goto Post999
    SET _TPROF_RUN_TYPE=-p
    goto ParseContinue
:Post999
    if "%_TPROF_RUN_TYPE%" == "-p" goto ParseContinue
    echo.
    echo run.tprof: %1 not allowed with %_TPROF_RUN_TYPE% option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -s option
Rem **
:HandleS
    shift
    if "%1" == "" goto S999
    set _TPROF_START_DELAY=%1
    goto ParseContinue
:S999
    echo.
```

```
    echo run.tprof: Missing time delay with -s option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -r option
Rem **
:HandleR
    shift

    if "%1" == "" goto R999
    set _TPROF_RUN_TIME=%1
    goto ParseContinue
:R999
    echo.
    echo run.tprof: Missing run time with -r option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -b option
Rem **
:HandleB
    shift
    if "%1" == "" goto B999
    set _TPROF_BUFSIZE=%1
    goto ParseContinue
:B999
    echo.
    echo run.tprof: Missing buffer size with -b option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -m option
Rem **
:HandleM
    shift
    if "%1" == "" goto M999
    if "%1" == "time" goto M100
    if "%1" == "Time" goto M100
    if "%1" == "TIME" goto M100
    if "%1" == "event" goto M100
    if "%1" == "Event" goto M100
    if "%1" == "EVENT" goto M100
    echo.
    echo run.tprof: Invaid mode with -m option. Must be TIME or EVENT. Quitting.
    echo.
    goto Exit
:M100
    set _TPROF_MODE=%1
    goto ParseContinue
:M999
    echo.
    echo run.tprof: Missing mode with -m option. Quitting.
    echo.
    goto Exit
```

```
Rem **
Rem ** Handle -e option
Rem **
:HandleE
    shift
    if "%1" == "" goto E999
    set _TPROF_EVENT=%1
    goto ParseContinue
:E999
    echo.
    echo run.tprof: Missing event name with -e option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Handle -c option
Rem **
:HandleC

    shift
    if "%1" == "" goto C999
    set _TPROF_CNT=%1
    goto ParseContinue
:C999
    echo.
    echo run.tprof: Missing count with -c option. Quitting.
    echo.
    goto Exit

Rem **
Rem ** Done parsing
Rem **
:DoneParsing
  set _TPROF_POST_RC=0

Rem ********************************************************************
Rem ********************************************************************
Rem                       Set environment
Rem ********************************************************************
Rem ********************************************************************

Rem **
Rem ** Try running TOOLSENV.CMD and hope it can be found along PATH.
Rem ** It could be found in the current directory if the user has
Rem ** added it to PATH.
Rem **
  echo run.tprof: Attempting to run TOOLSENV.CMD (along PATH) ...
  call TOOLSENV.CMD > NUL 2>NUL
  if not "%IBMPERF_TOOLS_PATH%" == "" goto Start
  echo run.tprof: ... either not found or didn't at least set IBMPERF_TOOLS_PATH
  echo.

Rem **
Rem ** Not found along PATH ...
Rem ** Now try running TOOLSENV.CMD from the current directory.
Rem **
```

```
   echo.
   if not exist .\TOOLSENV.CMD goto TryBin
   echo run.tprof: Attempting to run TOOLSENV.CMD (current directory) ...
   call .\TOOLSENV.CMD > NUL 2>NUL
   if not "%IBMPERF_TOOLS_PATH%" == "" goto Start
   echo run.tprof: ... either not found or didn't at least set IBMPERF_TOOLS_PATH

:TryBin
Rem **
Rem ** Not in the current directory or didn't set IBMPERF_TOOLS_PATH ...
Rem ** Now check if TOOLSENV.CMD is in 'bin' directory, in case user is
Rem ** running from the tools root directory.
Rem **
   if not exist bin\TOOLSENV.CMD goto GiveUp
   cd bin
   echo.
   echo run.tprof: Attempting to run bin\TOOLSENV.CMD (bin subdirectory) ...
   call TOOLSENV.CMD > NUL 2>NUL
   if not "%IBMPERF_TOOLS_PATH%" == "" goto Start
   echo run.tprof: ... either not found or didn't at least set IBMPERF_TOOLS_PATH

:GiveUp
Rem **
Rem ** TOOLSENV.CMD nowhere to be found *OR* it was found but it did

Rem ** not set IBMPERF_TOOLS_PATH.  Either way it's an error.
Rem **
   echo.
   echo run.tprof: Environment variable IBMPERF_TOOLS_PATH is not set.
   echo.
   echo Either no copy of TOOLSENV.CMD was found or IBMPERF_TOOLS_PATH has
   echo not been set.  You must either:
   echo - Change to the 'bin' subdirectory in the directory where the
   echo   tools were installed and run run.tprof from there, or
   echo - Copy TOOLSENV.CMD from the 'bin' subdirectory in the directory
   echo   where the tools were installed to this directory and then
   echo   re-run run.tprof.
   echo In any case, make sure TOOLSENV.CMD has been updated correctly if
   echo you haven't already done so.
   echo If you don't have a TOOLSENV.CMD run TINSTALL and one will be
   echo generated for you.
   goto Exit


:Start
Rem **
Rem **  Set defaults
Rem **
   echo.
   if "%IBMPERF_DATA_PATH%" == ""  set IBMPERF_DATA_PATH=%IBMPERF_TOOLS_PATH%
   if "%IBMPERF_JITA2N_PATH%" == "" set IBMPERF_JITA2N_PATH=%IBMPERF_DATA_PATH%
   if "%IBMPERF_JITA2N_NAME_ROOT%" == "" set IBMPERF_JITA2N_NAME_ROOT=log
   if "%IBMPERF_TPROF_REPORT_FILENAME%" == "" set IBMPERF_TPROF_REPORT_FILENAME=tprof.out

   if "%IBMPERF_TPROF_MODE%" == "" set IBMPERF_TPROF_MODE=TIME
   if not "%_TPROF_MODE%" == "" set IBMPERF_TPROF_MODE=%_TPROF_MODE%
```

```
   if "%IBMPERF_TPROF_TICK_RATE%" == "" set IBMPERF_TPROF_TICK_RATE=128
   if "%IBMPERF_TPROF_EVENT%" == "" set IBMPERF_TPROF_EVENT=INSTR
   if "%IBMPERF_TPROF_EVENTCNT%" == "" set IBMPERF_TPROF_EVENT=10000000

   if not "%_TPROF_EVENT%" == "" set IBMPERF_TPROF_EVENT=%_TPROF_EVENT%

   if "%IBMPERF_TPROF_MODE%" == "time" set IBMPERF_TPROF_MODE=TIME
   if "%IBMPERF_TPROF_MODE%" == "Time" set IBMPERF_TPROF_MODE=TIME
   if "%IBMPERF_TPROF_MODE%" == "event" set IBMPERF_TPROF_MODE=EVENT
   if "%IBMPERF_TPROF_MODE%" == "Event" set IBMPERF_TPROF_MODE=EVENT

   if "%IBMPERF_TPROF_MODE%" == "TIME" (
       if not "%_TPROF_CNT%" == "" set IBMPERF_TPROF_TICK_RATE=%_TPROF_CNT%
   ) else (
       if not "%_TPROF_CNT%" == "" set IBMPERF_TPROF_EVENTCNT=%_TPROF_CNT%
   )

   if "%IBMPERF_TPROF_MAJOR_CODES%" == "" set IBMPERF_TPROF_MAJOR_CODES=16 25

   if "%IBMPERF_TPROF_BUFFER_SIZE%" == "" set IBMPERF_TPROF_BUFFER_SIZE=3
   if not "%_TPROF_BUFSIZE%" == "" set IBMPERF_TPROF_BUFFER_SIZE=%_TPROF_BUFSIZE%

   set IBMPERF_TPROF_POST_OPTIONS=-off -clip 0 -show -a2nrdup -a2nmmi **MMI**
   if "%IBMPERF_SCREEN_LINES%" == "" set IBMPERF_SCREEN_LINES=0
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "yes" set IBMPERF_REMOVE_JITA2N_FILES=YES
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "YES" set IBMPERF_REMOVE_JITA2N_FILES=YES
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "no" set IBMPERF_REMOVE_JITA2N_FILES=NO
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "NO" set IBMPERF_REMOVE_JITA2N_FILES=NO
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "" set IBMPERF_REMOVE_JITA2N_FILES=YES
   if "%_LP%" == "" set _LP=%IBMPERF_JITA2N_PATH%\%IBMPERF_JITA2N_NAME_ROOT%



Rem **
Rem **  Change screen size if required
Rem **
   if %IBMPERF_SCREEN_LINES% EQU 0 goto NoSizeChange
   @echo on
   MODE CON: LINES=%IBMPERF_SCREEN_LINES%
   @echo off
:NoSizeChange

Rem ********************************************************************
Rem ********************************************************************
Rem                      Display environment
Rem ********************************************************************
Rem ********************************************************************

Rem **
Rem **  Display what we're using
Rem **
   echo.
   echo IBMPERF_TOOLS_PATH           = %IBMPERF_TOOLS_PATH%
   echo IBMPERF_DATA_PATH            = %IBMPERF_DATA_PATH%
   echo IBMPERF_SYMBOLS_PATH         = %IBMPERF_SYMBOLS_PATH%
   echo IBMPERF_TPROF_BUFFER_SIZE    = %IBMPERF_TPROF_BUFFER_SIZE%
```

```
   echo IBMPERF_TPROF_MODE            = %IBMPERF_TPROF_MODE%
   if "%IBMPERF_TPROF_MODE%" == "TIME" (
       echo IBMPERF_TPROF_TICK_RATE       = %IBMPERF_TPROF_TICK_RATE%
   ) else (
       echo IBMPERF_TPROF_EVENT           = %IBMPERF_TPROF_EVENT%
       echo IBMPERF_TPROF_EVENTCNT        = %IBMPERF_TPROF_EVENTCNT%
   )
   echo IBMPERF_TPROF_MAJOR_CODES     = %IBMPERF_TPROF_MAJOR_CODES%
   echo IBMPERF_TPROF_POST_OPTIONS    = %IBMPERF_TPROF_POST_OPTIONS%
   echo IBMPERF_TPROF_REPORT_FILENAME  = %IBMPERF_TPROF_REPORT_FILENAME%
   echo IBMPERF_JITA2N_PATH           = %IBMPERF_JITA2N_PATH%
   echo IBMPERF_JITA2N_NAME_ROOT      = %IBMPERF_JITA2N_NAME_ROOT%
   echo IBMPERF_REMOVE_JITA2N_FILES   = %IBMPERF_REMOVE_JITA2N_FILES%
   echo IBMPERF_SCREEN_LINES          = %IBMPERF_SCREEN_LINES%
   if not "%_TPROF_START_DELAY%" == "" echo TPROF Start Delay            = %_TPROF_START_DELAY%
Seconds
   if not "%_TPROF_RUN_TIME%" == "" echo TPROF Run Time               = %_TPROF_RUN_TIME% Seconds
   echo Current Directory             = %CD%
   echo Current Date and Time         = %DATE% at %TIME%
   if "%_TPROF_RUN%" == "trace" echo ***** Running trace portion only *****
   if "%_TPROF_RUN%" == "post" echo ***** Running post-processing portion only *****
   echo.

Rem ********************************************************************
Rem ********************************************************************
Rem                        Do the work
Rem ********************************************************************
Rem ********************************************************************

Rem **
Rem **  If they only want to post-process then go do that
Rem **
   if "%_TPROF_RUN_TYPE%" == "-p" goto SwtraceOff

:Trace
Rem **
Rem **  Get ready to trace ...
Rem **
   echo.
   echo **
   echo ** Initializing SWTRACE ...
   echo **
   @echo on
   SWTRACE INIT /S %IBMPERF_TPROF_BUFFER_SIZE%
   @echo off
   if not "%errorlevel%" == "99" goto Continue100
   echo.
   echo Looks like PERFDD.SYS isn't loaded.
   echo Did you run \"TINSTALL\" successfully?
   goto Exit
   if not "%errorlevel%" == "0" goto SwtraceError

:Continue100
   if "%IBMPERF_TPROF_MODE%" == "EVENT"  goto SetEvent
   if "%IBMPERF_TPROF_MODE%" == "event"  goto SetEvent
   if "%IBMPERF_TPROF_MODE%" == "Event"  goto SetEvent
```

```
Rem **
Rem ** Set up for TIME-based profiling
Rem **
   @echo on
   SWTRACE SETRATE %IBMPERF_TPROF_TICK_RATE%
   @if not "%errorlevel%" == "0" @goto SwtraceError
   goto Continue110

:SetEvent
Rem **
Rem ** Set up for EVENT-based profiling
Rem **
   @echo on
   SWTRACE EVENT %IBMPERF_TPROF_EVENT% -c %IBMPERF_TPROF_EVENTCNT%
   @if not "%errorlevel%" == "0" @goto SwtraceError

:Continue110
   @echo on
   SWTRACE ENABLE %IBMPERF_TPROF_MAJOR_CODES%
   @if not "%errorlevel%" == "0" @goto SwtraceError
   @echo off

Rem **
Rem ** Delete jita2n* and jtnm* files if needed
Rem **
   if "%IBMPERF_REMOVE_JITA2N_FILES%" == "NO" goto Continue120
   @echo on
   ERASE %_LP%-jita2n* > NUL 2>NUL
   ERASE %_LP%-jtnm* > NUL 2>NUL
   @echo off

:Continue120

   @echo off
Rem **
Rem ** Remove other log* files if needed
Rem **
   if "%IBMPERF_JPROF_LOG_FILES_TO_REMOVE%" == "" goto Continue125
   @echo on
   ERASE %IBMPERF_JPROF_LOG_FILES_TO_REMOVE% > NUL 2>NUL
   @echo off

:Continue125
Rem **
Rem ** About to start tracing ...
Rem **
   @echo off
   if "%_TPROF_START_DELAY%" == "" goto StartPrompt
   echo.
   echo *****
   echo ***** Will automatically start tracing in %_TPROF_START_DELAY% seconds ...
   echo *****
   if not "%_TPROF_START_DELAY%" == "0" ASK /z %_TPROF_START_DELAY%
   echo.
   echo Trace started at %TIME% ...
   goto SwtraceOn
```

```
:StartPrompt
   @echo off
   echo.
   echo ********************************************************************
   echo Get your application loaded and running ...
   echo.
   echo If TPROFing a java application you must invoke java using the '-Xrun'
   echo option.  This allows jitted method information to be obtained.
   echo.
   echo Invoke java as follows:
   echo   java -Xrunjprof:jita2n,threadinfo,fnm=%_LP%,pidx AppName
   echo.
   echo ********************************************************************
   echo.

   echo *****
   echo *****
   @rem ASK /w ***** Press ENTER when ready to begin tracing or Ctrl-C to quit ...
   echo.
   echo Trace started at %TIME% ...

:SwtraceOn
   @echo on
   SWTRACE ON
   @if not "%errorlevel%" == "0" goto SwtraceError
   @echo off

Rem ******************************************
Rem *                                        *
Rem *       User is now tracing ...          *
Rem *                                        *
Rem ******************************************

   @echo ##_done_start_tools##

   if "%_TPROF_RUN_TIME%" == "" goto ExitStart
   echo.
   echo *****
   echo ***** Will automatically stop tracing in %_TPROF_RUN_TIME% seconds ...
   echo *****
   echo.
   ASK /z %_TPROF_RUN_TIME%
   goto SwtraceOff

:StopPrompt
   echo.
   echo *****
   echo *****
   ASK /w ***** Tracing. Press ENTER when ready to stop ...

:SwtraceOff
   @echo on
   SWTRACE OFF
   @if not "%errorlevel%" == "0" goto SwtraceError
   @echo off

   echo.
```

```
   echo **
   echo ** Collecting trace ...
   echo **
   echo.
   echo Trace ended at %TIME% ...

   @echo on
   SWTRACE GET %IBMPERF_DATA_PATH%\swtrace.nrm2
   @if not "%errorlevel%" == "0" goto SwtraceError
   SWTRACE DISABLE
   SWTRACE FREE
   @echo off

Rem **
Rem **  If they only wanted to trace then we're done
Rem **
   if "%_TPROF_RUN_TYPE%" == "-t" goto Continue160

Rem **
Rem **  This is where post-processing begins ...
Rem **
:PostProcess
   SET _TPROF_POST_JDIR=-jdir %_LP%
   if exist %_LP%-jita2n* goto Continue140
   if exist %_LP%-jtnm* goto Continue140
   echo.
   echo No jita2n* or jtnm* file(s) found. Assuming no java ...
   SET _TPROF_POST_JDIR=

:Continue140
   @echo off
   echo.
   echo **
   echo ** Generating TPROF report ...
   echo **

   if "%IBMPERF_SYMBOLS_PATH%" == "" goto Continue145
   @echo on
   SET A2N_SEARCH_PATH=%IBMPERF_SYMBOLS_PATH%
   @echo off

:Continue145
   @echo on
   @echo IBMPERF_DATA_PATH:  %IBMPERF_DATA_PATH%
   @echo IBMPERF_TOOLS_PATH: %IBMPERF_TOOLS_PATH%
   POST -off -r %IBMPERF_DATA_PATH%\swtrace.nrm2 %_TPROF_POST_JDIR%
%IBMPERF_TPROF_POST_OPTIONS%
   @set _TPROF_POST_RC=%errorlevel%
   @CALL mergetprof.cmd tprof_e.out
   @rem if "%IBMPERF_DATA_PATH%" == "%CD%" @goto Continue150
   @rem MOVE /Y tprof.out %IBMPERF_DATA_PATH%\%IBMPERF_TPROF_REPORT_FILENAME%
   @rem MOVE /Y tprof_e.out %IBMPERF_DATA_PATH%\%IBMPERF_TPROF_REPORT_FILENAME%
   @rem **
   @rem ** Copy all the other post junk to the data directory
   @rem **
   @rem MOVE /Y a2n.err  %IBMPERF_DATA_PATH%\. > NUL 2>NUL
   @rem @MOVE /Y a2n.mod  %IBMPERF_DATA_PATH%\. > NUL 2>NUL
```

```
   @rem @MOVE /Y a2n.proc %IBMPERF_DATA_PATH%\. > NUL 2>NUL
   @rem @MOVE /Y post.msg %IBMPERF_DATA_PATH%\. > NUL 2>NUL
   @rem @MOVE /Y ptree    %IBMPERF_DATA_PATH%\. > NUL 2>NUL
   @rem @MOVE /Y dbfile   %IBMPERF_DATA_PATH%\. > NUL 2>NUL
   @rem @MOVE /Y arc      %IBMPERF_DATA_PATH%\. > NUL 2>NUL

:Continue150
   @echo off
Rem **
Rem **  Got here because were were doing -a or -p
Rem **
   if not "%_TPROF_POST_RC%" == "0" goto PostError
   echo.
   echo ********************************************************************
   echo   TProf report "%IBMPERF_DATA_PATH%\%IBMPERF_TPROF_REPORT_FILENAME%" generated.
   echo ********************************************************************
   @echo ##_done_get_the_file##
   goto Exit

:Continue160
   @echo off
Rem **
Rem **  Got here because were were doing -t
Rem **
   echo.
   echo ********************************************************************
   echo   Trace file "%IBMPERF_DATA_PATH%\swtrace.nrm2" generated.
   echo ********************************************************************
   goto Exit

:PostError
   @echo off
   echo.
   echo **ERROR** **ERROR** **ERROR**
   echo   POST command error. TProf report may not have been generated.
   echo.
   echo   - Look for report file "%IBMPERF_DATA_PATH%\%IBMPERF_TPROF_REPORT_FILENAME%".
   echo   - Go over all screen output.
   echo   - Look in file "%IBMPERF_DATA_PATH%\post.msg" for addtional
   echo     error information.
   echo **ERROR** **ERROR** **ERROR**
   echo.
   goto Exit

:SwtraceError
   @echo off
   echo.
   echo **ERROR** **ERROR** **ERROR**
   echo SWTRACE command error.  Quitting.
   echo **ERROR** **ERROR** **ERROR**
   echo.
   goto Exit

:Exit
   set _TPROF_START_DELAY=
   set _TPROF_RUN_TIME=
   set _TPROF_RUN_TYPE=
```

```
   set _TPROF_POST_JDIR=
   set _TPROF_BUFSIZE=
   set _TPROF_MODE=
   set _TPROF_EVENT=
   set _TPROF_CNT=
   set _TPROF_POST_RC=

   @echo ##_done_stop_tools##

:ExitStart
@endlocal
```