

# STROBE MVS

---

## STROBE Java Feature

Release 3.0



**COMPUWARE®**

Please direct questions about STROBE MVS  
or comments on this document to:

**STROBE MVS Technical Support**

Compuware Corporation  
124 Mount Auburn Street  
Cambridge MA 02138-5758  
**1-800-585-2802 or**  
**1-617-661-3020**  
**1-617-498-4010 (fax)**  
strobe-sup@compuware.com

Outside the USA and Canada, please contact  
your local Compuware office or agent.

This document and the product referenced in it are subject to the following legends:

Copyright 2002 Compuware Corporation. All rights reserved. Unpublished rights reserved under the Copyright Laws of the United States.

U.S. GOVERNMENT RIGHTS-Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in Compuware Corporation license agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable. Compuware Corporation.

This product contains confidential information and trade secrets of Compuware Corporation. Use, disclosure, or reproduction is prohibited without the prior express written permission of Compuware Corporation. Access is limited to authorized users. Use of this product is subject to the terms and conditions of the user's License Agreement with Compuware Corporation.

STROBE, iSTROBE, and APMpower are trademarks or registered trademarks of Compuware Corporation.

AD/Cycle, BookManager, CICS, DB2, IBM, IMS/ESA, Language Environment, MQSeries, OS/2, Software Mall, VisualGen, and VTAM are trademarks of International Business Machines Corporation. Microsoft is a registered trademark of Microsoft Corporation. Windows, Windows NT, and Windows 98 are trademarks of Microsoft Corporation. Attachmate and EXTRA! are registered trademarks of Attachmate Corporation. RUMBA is a registered trademark of Wall Data Inc. WRQ and Reflection are registered trademarks of WRQ, Inc. Java and Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Adobe ® Acrobat ® Reader copyright © 1987-2001 Adobe Systems Incorporated. All rights reserved. Adobe and Acrobat are trademarks of Adobe Systems Incorporated.

All other company and product names are trademarks or registered trademarks of their respective owners.

# Contents

<b>Figures</b> .....	<b>v</b>
<b>Introduction</b> .....	<b>ix</b>
How This Manual Is Organized .....	ix
How to Use This Manual .....	ix
The STROBE Library .....	ix
STROBE Feature Manuals .....	x
Online Documentation .....	x
Online Help .....	x
Other Compuware Application Performance Management Products .....	xi
iSTROBE .....	xi
SQL Analysis Feature .....	xi
APMpower .....	xi
Compuware APM Technical Support .....	xi
Compuware APM Training .....	xi
Compuware APM Service Offerings .....	xii
APM Installation Assurance .....	xii
Application Performance Management Consulting .....	xii
Application Performance Assessment .....	xii
<b>Chapter 1. Overview</b> .....	<b>1-1</b>
Benefits of the STROBE Java Feature .....	1-1
Concepts and Terminology .....	1-2
Overview of the JVM Java Target Environment .....	1-3
Measuring JVM and CICS Applications .....	1-4
Measuring JVM and DB2 Applications .....	1-4
Overview of the HPJ Java Target Environment .....	1-5
Where to Find More Information .....	1-6
<b>Chapter 2. Using the STROBE Java Feature</b> .....	<b>2-1</b>
Requirements for the STROBE Java Feature .....	2-1
JVM Java Requirements .....	2-1
HPJ Java Requirements .....	2-1
Measuring the Application .....	2-2
Measuring an Active Job .....	2-2
Measuring a Job That is not yet Executing .....	2-2
Configuring Java Data Collection .....	2-3
Java Targeting and Reporting .....	2-4
Using Java Targeting .....	2-4
Targeting JVM Java Code for Measurement .....	2-5
Specifying Target Information .....	2-5
Using the STROBE Java Target Selector .....	2-6
Creating a Performance Profile .....	2-8
Controlling the Level of Detail in the Reports .....	2-8
Tailoring the STROBE HPJ Java Feature Reports .....	2-8
<b>Chapter 3. Analyzing a HPJ Java Performance Profile</b> .....	<b>3-1</b>
Finding Significant Activity in the Java Application .....	3-1
Java Class Summary Report .....	3-1
Attribution of CPU Execution Time Report .....	3-3

<b>Chapter 4. Analyzing a JVM Java Performance Profile</b> .....	<b>4-1</b>
Measurement Session Data Report .....	4-1
Java Targeting Information Report .....	4-2
Java Environment Report .....	4-3
Java CPU Reports .....	4-3
Java CPU Usage by Called Method Summary Report .....	4-3
Java CPU Usage by Called Method Report .....	4-4
The iSTROBE Java Activity by Called Method Report .....	4-7
Java CPU by Executing Method Report .....	4-8
Java Wait Time Reports .....	4-10
Java Wait Time by Called Method Summary Report .....	4-10
Java Wait Time by Called Method Report .....	4-11
Java Wait Time by Method Report .....	4-12
Program Section Usage Summary Report .....	4-13
Program Usage by Procedure Report .....	4-15
Attribution Reports .....	4-15
Attribution of CPU Execution Time Report .....	4-15
Attribution of CPU Wait Time Report .....	4-16
<b>Index</b> .....	<b>I-1</b>

# Figures

1-1.	Overview of STROBE Tasks .....	1-3
1-2.	Java Virtual Machine Architecture .....	1-4
1-3.	Overview of Java Execution in a CICS TS 1.3 Environment .....	1-5
2-1.	STROBE - DATA COLLECTORS Panel.....	2-3
2-2.	STROBE - JAVA TARGETING Panel .....	2-5
2-3.	STROBE - JAVA TARGETING EXTENDED EDIT Panel .....	2-6
2-4.	STROBE - JAVA TARGETING LIST Panel .....	2-7
2-5.	STROBE - TAILOR REPORTS Panel .....	2-9
3-1.	Java Class Summary Report .....	3-2
3-2.	CPU Usage by Java Method Report .....	3-3
3-3.	Attribution of CPU Execution Time Report .....	3-4
4-1.	Measurement Session Data Report .....	4-2
4-2.	Java Targeting Information Report.....	4-2
4-3.	Java Environment Report .....	4-3
4-4.	Java CPU Usage by Called Method Summary Report .....	4-4
4-5.	Java CPU Usage by Called Method Report .....	4-5
4-6.	iSTROBE Java Activity by Called Method Report .....	4-7
4-7.	iSTROBE Java Call Stacks Report .....	4-8
4-8.	Java CPU By Executing Method Report.....	4-9
4-9.	Java Wait Time by Called Method Summary Report .....	4-10
4-10.	Java Wait Time by Called Method Report.....	4-11
4-11.	Java Wait Time by Method Report .....	4-12
4-12.	Program Section Usage Summary Report .....	4-13
4-13.	Program Usage by Procedure Report .....	4-15
4-14.	Attribution of CPU Execution Time Report .....	4-16
4-15.	Attribution of CPU Wait Time Report.....	4-16



# Summary of Changes

This section lists the changes to the Java Feature for STROBE MVS for Sysplex Release 3.0.

---

## Changes to the Feature

In addition to supporting measurement of High Performance Java (HPJ) applications, STROBE now supports measurement of applications running under the Java Virtual Machine Version 1.3.1. CICS Transaction Server 2.2 is supported along with the measurement of both JDBC (dynamic) and SQLJ (static) DB2 applications. The following lists the changes to the STROBE Java Feature that enable you to measure and analyze JVM applications:

- New ISPF panels that allow you to specify Java targeting information. You can either identify the targets yourself, or you can use the STROBE Java targeting selector that lists the contents of the Java applications and then you can pick what you want to target.
- New summary reports that show CPU and wait time data for Java applications.
- New “called method” reports that show CPU and wait time Java call stack data for either the initial application method or the user-targeted method.
- New “executing method” reports that show CPU and wait time for the method STROBE found running when it took a sample. These reports distinguish between code that has been interpreted or that was processed by the Just-In-Time (JIT) compiler.
- New targeting and environment reports that show targeting data and JVM profile information.

---

## Changes to the Manual

The following changes have been made to this manual as a result of JVM support:

- Chapter 1
  - Overview section describing the JVM environment and its interface with CICS regions and DB2 applications.
- Chapter 2
  - New JVM Java targeting ISPF panel descriptions and examples and a section describing how targeting application methods for measurement impacts the Performance Profile report contents.
- Chapter 3
  - Descriptions and examples of HPJ reports.
- Chapter 4 (new)
  - Descriptions and examples of JVM reports, including a description of the benefits of using iSTROBE to analyze JVM application measurement data.



## Introduction

This manual describes measurement concepts applicable to and specific data made available by the STROBE Java Feature of the STROBE MVS Application Performance Measurement System. The STROBE Java Feature augments functions provided by the basic STROBE system.

The STROBE Application Performance Measurement System and the STROBE Java Feature are products designed for IBM MVS/ESA, IBM OS/390, and IBM z/OS systems. The STROBE Java Feature is designed for use with applications built with IBM's High Performance Java (HPJ) compiler packaged with VisualAge for Java for the OS/390 and the Java Virtual Machine (JVM) Version 1.3.1.

---

## How This Manual Is Organized

Chapter 1, "Overview" presents an overview of the STROBE Java Feature and how it interacts with the Java operating environments.

Chapter 2, "Using the STROBE Java Feature" explains how to use the STROBE Java Feature.

Chapter 3, "Analyzing a HPJ Java Performance Profile" describes the Performance Profile reports for a HPJ Java program.

Chapter 4, "Analyzing a JVM Java Performance Profile" describes the Performance Profile reports for a JVM Java program.

---

## How to Use This Manual

You should read Chapter 1, "Overview" and Chapter 2, "Using the STROBE Java Feature" before submitting a measurement request. To interpret a STROBE Performance Profile for HPJ Java applications, read Chapter 3, "Analyzing a HPJ Java Performance Profile". To interpret a STROBE Performance Profile for JVM Java applications, read Chapter 4, "Analyzing a JVM Performance Profile".

---

## The STROBE Library

The STROBE base product manuals include:

- *STROBE MVS Concepts and Facilities*, document number CWSTGX3A  
*STROBE MVS Concepts and Facilities* explains how to decide which programs and online regions to measure, when to measure them, and how to interpret the reports in the STROBE Performance Profile.
- *STROBE MVS Messages*, document number CWSTXM3A  
*STROBE MVS Messages* lists all messages and abnormal termination (ABEND) codes, describes how to interpret them, and in many cases suggests a corrective action.
- *STROBE MVS System Programmer's Guide*, document number CWSTXI3A  
The *STROBE MVS System Programmer's Guide* explains how to install and maintain STROBE.

- *STROBE MVS User's Guide*, document number CWSTUX3A and the *STROBE MVS User's Guide with Advanced Session Management*, document number CWSTUA3A

The *STROBE MVS User's Guide* explains how to use STROBE to measure application performance. The *STROBE MVS User's Guide with Advanced Session Management* explains how to use STROBE with the STROBE Advanced Session Management Feature to measure application performance. Users who have the STROBE Advanced Session Management Feature will use this manual rather than the *STROBE MVS User's Guide*.

- *STROBE MVS Application Performance Measurement System Quick Reference*

The *STROBE MVS Application Performance Measurement System Quick Reference* is a convenient reference for how to use STROBE and for interpreting the STROBE Performance Profile.

## STROBE Feature Manuals

These manuals describe the optional features of the STROBE MVS Application Performance Measurement System. Each manual describes measurement concepts applicable to and specific data made available by the feature.

- *STROBE MVS User's Guide with Advanced Session Management*, document number CWSTUA3A
- *STROBE ADABAS/NATURAL Feature*, document number CWSTUN3A
- *STROBE CA-IDMS Feature*, document number CWSTUR3A
- *STROBE CICS Feature*, document number CWSTUC3A
- *STROBE COOL:Gen Feature*, document number CWSTUG3A
- *STROBE CSP Feature*, document number CWSTUP3A
- *STROBE DB2 Feature*, document number CWSTUD3A
- *STROBE IMS Feature*, document number CWSTUI3A
- *STROBE Interface Feature*, document number CWSTUF3A
- *STROBE Java Feature*, document number CWSTUJ3A
- *STROBE MQSeries Feature*, document number CWSTUM3A
- *STROBE UNIX System Services Feature*, document number CWSTUU3A

## Online Documentation

STROBE manuals are available in HTML, Adobe Acrobat PDF format, and IBM BookManager format, on CD-ROM and at Compuware's technical support Web site at <http://frontline.compuware.com>.

---

## Online Help

STROBE products provide the following online information:

- STROBE/ISPF Online Tutorials, Option T from the STROBE/ISPF STROBE OPTIONS menu
- STROBE/ISPF Online Message Facility, Option M from the STROBE/ISPF STROBE OPTIONS menu

---

## Other Compuware Application Performance Management Products

The following products and features work in conjunction with STROBE MVS Application Performance Measurement System. These tools extend the benefits of application performance management (APM).

### iSTROBE

iSTROBE enables you to view and analyze STROBE Performance Profile data on a workstation using a standard Web browser. Easy to install and easy to use, iSTROBE guides you through the performance analysis process and offers recommendations for improving performance. iSTROBE simplifies the performance analysis of applications that you measure with STROBE. For more information on iSTROBE, see the *iSTROBE Getting Started Guide*.

### SQL Analysis Feature

The SQL Analysis Feature works in conjunction with STROBE and iSTROBE or APMpower to supply access path analyses and database and SQL coding recommendations for DB2 applications measured by STROBE. The SQL Analysis Feature pinpoints the most resource-consumptive static or dynamic SQL statements, explains why these statements might be inefficient, and provides recommendations to improve the performance of the DB2 application. For more information on the SQL Analysis Feature, see the *STROBE MVS User's Guide* or the *STROBE MVS User's Guide with Advanced Session Management*.

### APMpower

The APMpower Application Performance Analysis System extends the benefits of STROBE to application developers who use workstations to develop, test, and maintain MVS applications. Developers employ the APMpower graphical user interface and advanced analytical aids to navigate the Performance Profile, analyze and improve application performance, and share performance knowledge across the IS organization. For more information about APMpower, see the APMpower documentation.

---

## Compuware APM Technical Support

For North American customers, for technical support, please contact the Technical Support department by telephone at (800) 585-2802 or (617) 661-3020, by fax at (617) 498-4010, or by e-mail at [strobe-sup@compuware.com](mailto:strobe-sup@compuware.com).

To access online technical support, visit Compuware's FrontLine page on the World Wide Web at <http://frontline.compuware.com> and select the product "STROBE and APMpower."

For other international customers, please contact your local Compuware office or STROBE supplier.

---

## Compuware APM Training

Compuware's Education Resources Group offers a range of training options for organizations that use STROBE, iSTROBE, and APMpower. To arrange Application Performance Management training, please contact Compuware at 1-800-835-3190 or visit Compuware's Education Resources Group at <http://www.compuware.com/training>

For other international customers, please contact your local Compuware office or STROBE supplier for a complete list of APM Training offerings.

---

## Compuware APM Service Offerings

For North American customers, for information about current service offerings, please contact your local Compuware sales office or call Compuware Corporate Headquarters at 1-800-COMPUWARE (266-7892) or visit Compuware's APM Product page on the World Wide Web at <http://www.compuware.com/products/strobe>.

For other international customers, please contact your local Compuware office or STROBE supplier for a complete list of Services offerings.

### APM Installation Assurance

The APM Installation Assurance service assists you in planning for, installing, customizing and using APM products. The service will help you maximize the value and benefits derived from the APM product family.

Consulting engineers work closely with your IT personnel to understand your operating environment and your organization's APM goals. The engineer will assist you in developing a customization and installation plan for STROBE, iSTROBE, and APMpower. The engineer will oversee the installation process and verify product readiness. The engineer will also help set up measurement request schedules, request groups, history records, AutoSTROBE measurement requests, and will verify the installation of the SQL Analysis Feature.

With APM Installation Assurance services, your organization can immediately maximize the value received from your investment in the APM product family. You will also benefit from a fully customized installation that will enhance the product functionality and increase the automation aspects of your APM initiatives.

### Application Performance Management Consulting

The Application Performance Management (APM) Consulting services assist you in identifying and resolving specific performance problems in your OS/390 business-critical applications.

Using STROBE, iSTROBE, and APMpower, consulting engineers work closely with your IT personnel to measure an application's performance, identify performance improvement opportunities and make recommendations for implementing solutions.

With APM Consulting services, your organization cannot only resolve problems quickly and effectively, but also gain the skills necessary to prevent application performance degradation in the future.

### Application Performance Assessment

The Application Performance Assessment (APA) service assists you in achieving a higher level of performance for your OS/390 business-critical applications.

Using STROBE, iSTROBE, and APMpower, consulting engineers work closely with your IT personnel to evaluate the efficiency of business-critical applications, identify opportunities for improving performance and document the potential savings that can result from implementing recommended solutions.

With APA services, you cannot only improve application performance quickly and effectively, but also gain the knowledge and skills necessary to implement and sustain a process-oriented application performance management (APM) program.

# Chapter 1.

## Overview

The STROBE MVS Application Performance Measurement System is a product that determines where and how time is spent in online regions and batch processing programs. You can produce a collection of reports with STROBE that helps you determine how to revise applications to improve their performance.

The STROBE Java Feature provides detailed performance information when measuring:

- Java class files executing under JVM program objects running in a batch OMVS system, or in a CICS Transaction Server 2.2. and higher environment. To obtain JVM measurement data, installation of the STROBE UNIX System Services Feature is a requirement. To obtain Java measurement data for a CICS environment, the STROBE CICS Feature is a requirement.
- HPJ program objects running in a batch OMVS system, or in a CICS Transaction Server 1.3 and higher environment. These applications are built with IBM's High Performance Java compiler packaged with IBM's VisualAge for Java for the OS/390 operating system.

This chapter describes the benefits of the STROBE Java Feature, outlines concepts and terminology of the Feature, and provides an overview of the Java environment.

---

## Benefits of the STROBE Java Feature

The STROBE Java Feature collects application performance information as the application executes. When measurement completes, STROBE organizes this information into the STROBE *Performance Profile*, a series of reports that show where and how time is spent during application execution, pinpointing possible areas for performance improvement. In the JVM Environment, The Performance Profile provides:

- a summary of the CLASSPATH information pertaining to the JVM environment that STROBE measures
- a summary of execution and wait activity at the method level
- execution and wait information for the executing method as far STROBE can trace back the original invoker and all calls made immediately as a result of the call by the original invoker
- execution and wait information for the method or service routine activity that was processing at the time STROBE took a measurement sample.
- CPU and wait attribution report with Java information.
- Program Section Usage Summary and Program Usage by Procedure reports with Java data.
- wait attribution for the OMVS and BPXBATCH environments
- In a CICS region, the STROBE CICS Feature Transaction Profiling reports show Java execution and wait delay information.

In the HPJ Environment, The Performance Profile provides

- a summary of CPU use in Java classes

- execution and attribution activity specified by offset within Java modules, classes, or methods
- the Java module, class, method and return address that invoked a system service routine
- wait attribution for BPXBATCH environment

The STROBE Java Feature helps you to measure, analyze, and improve the performance of Java programs by providing detailed performance information for Java programs that run in batch environments, that run in a CICS Transaction Server 1.3 and higher environment and in a OS/390 UNIX System Services environment. This enables you to develop and maintain more efficient and responsive applications throughout the application life cycle.

The next section discusses key concepts and terms that are central to the use of the STROBE Java Feature.

---

## Concepts and Terminology

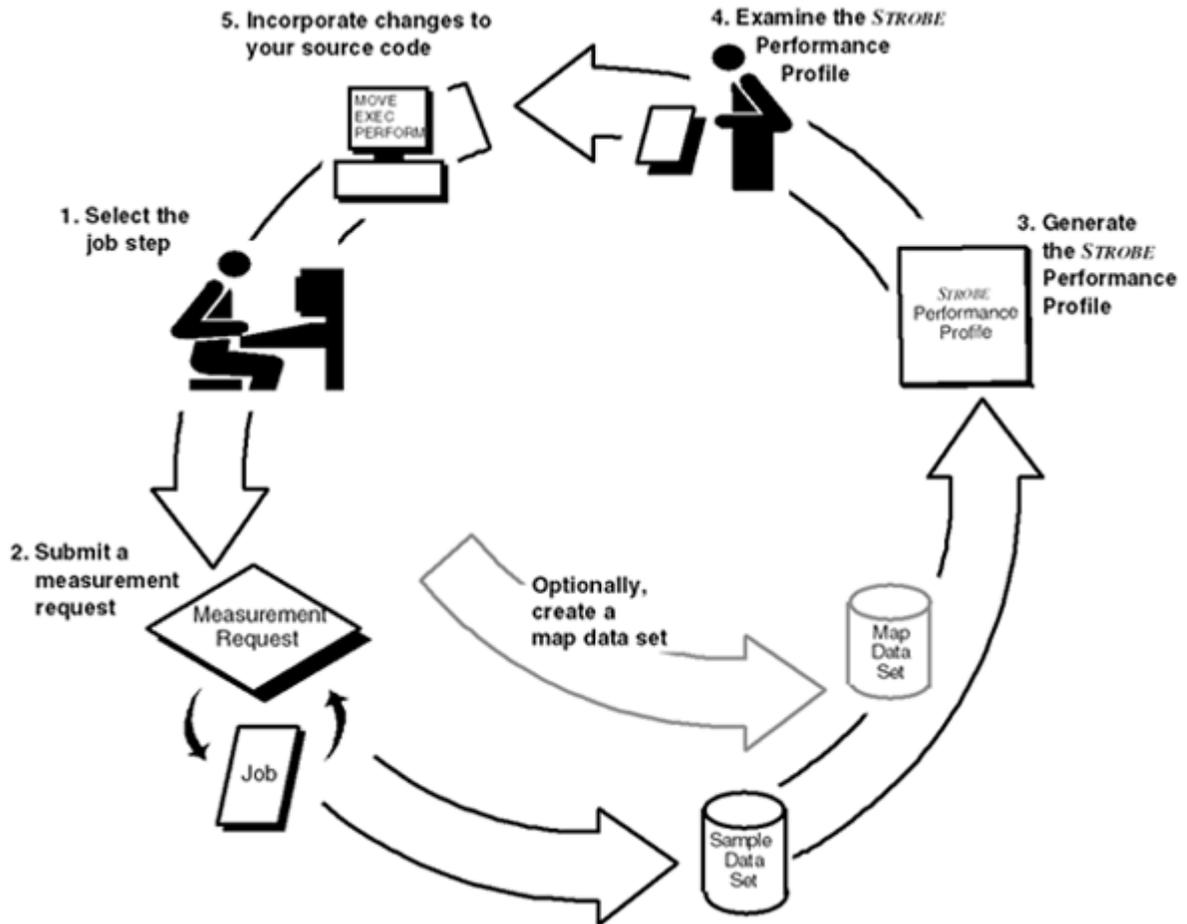
Before using the STROBE Java Feature, it is helpful to become familiar with the basic concepts and terminology specific to STROBE and the STROBE Java Feature. The following sections introduce you to these concepts and terms.

A *measurement request* specifies the parameters for measuring the performance of an application. When the application is active, STROBE begins a *measurement session*, an interval where it collects performance data about the application while it is executing. STROBE stores measurement data in a *sample data set*, a file that contains the information collected during a single measurement session. Each measurement session corresponds to one sample data set.

After STROBE closes the sample data set, you can use it to create the *Performance Profile*, a hierarchical series of reports that present the performance data collected during a measurement session. These reports show where and how the application spends its time during execution.

*Attribution* identifies the methods that invoke service routines causing CPU time. Review the Attribution of CPU Execution Time report when Java methods are responsible for significant CPU use.

Figure 1-1. Overview of STROBE Tasks



The STROBE Java Feature provides specific information on the Java programs that it measures. Java is an object-oriented programming language. In object-oriented programming, *objects* represent data. Objects have two sections, fields (instance variables) and methods. A *field* determines what an object is. A *method* determines what an object does. A Java *class* is a collection of fields and methods that encapsulate functionality into a reusable and dynamically loadable object.

STROBE provides information that enables you to identify a Java program's resource consumption in a class and, within the class, in a method.

---

## Overview of the JVM Java Target Environment

The Java Virtual Machine is the runtime instance in which a Java application executes. When the Java bytecode is compiled, it can execute on the Java Virtual Machine (JVM). The JVM defines a set of specifications that enable Java programs to run on virtually any platform.

Figure 1-2 shows an example of the Java Virtual Machine and the process that takes place from source code to execution. The first part of the process is the compilation of the Java source code to Java bytecode. Then the JVM starts and takes the name of the class for the .class file. It is then loaded by the class loader which processes the bytecode stream and sends it to the byte code verifier.

The structure and the bytecode validity is checked and if it is in order, it is sent on to either the JIT compiler or the Java interpreter. The JIT compiler will compile the byte code to machine code and the method, when invoked, will be executing machine code. If the interpreter receives the byte code, it executes the code, and it is either processed by the hardware or trapped by the operating system and indirectly executed.

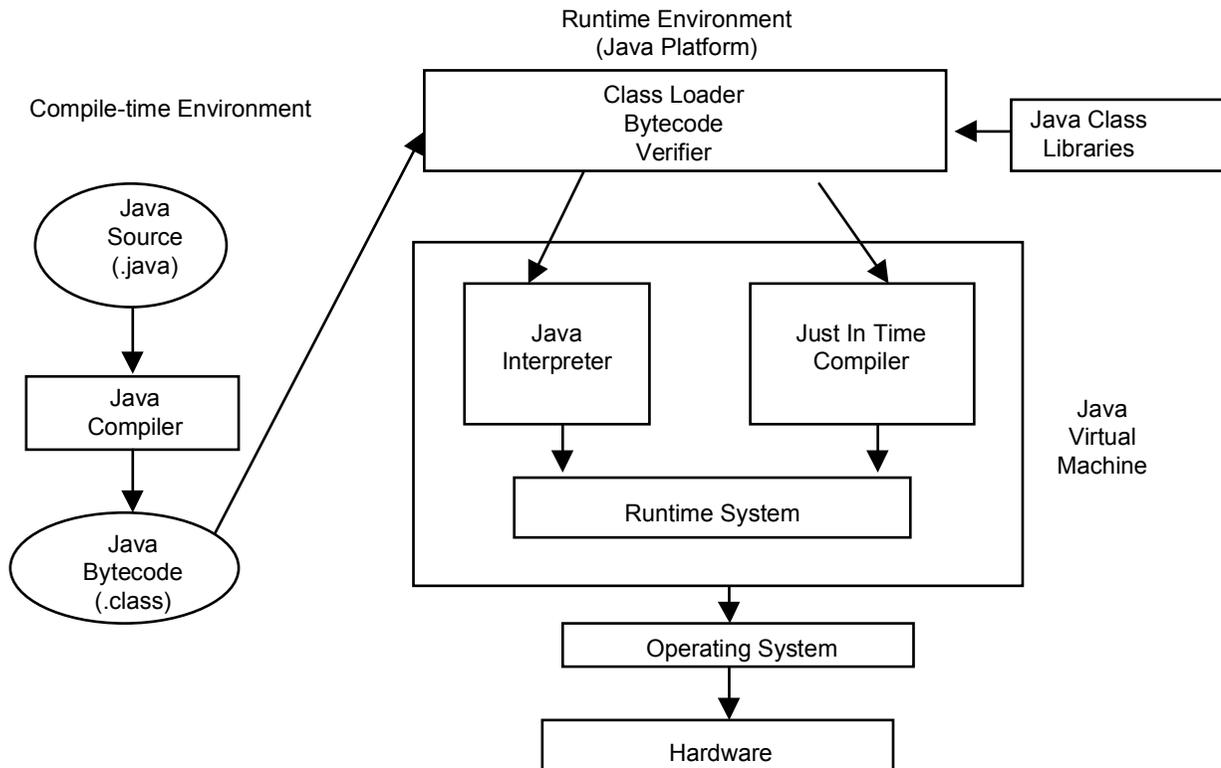
## Measuring JVM and CICS Applications

If the JVM is running in a CICS region, it has been optimized for the execution of CICS transactions. CICS manages a pool of JVMs. A JVM may be in use or available for reuse. Characteristics of the JVM are specified in JVM profiles that are defined for a CICS region. The profile contains pertinent information, including the name of the JVM properties file. See the *IBM CICS Transaction Server for z/OS: Java Applications in CICS* documentation for more information about CICS and the JVM.

## Measuring JVM and DB2 Applications

A JVM Java program can access DB2 through the JDBC and SQLJ application programming interfaces (APIs). The JDBC API provides an interface for Java programs to process relational data, but only for dynamic SQL. If the SQL uses static execution, the SQLJ API is used. See the IBM documentation covering the JDBC and SQLJ APIs for information about Java and the DB2 environment.

Figure 1-2. Java Virtual Machine Architecture



## Overview of the HPJ Java Target Environment

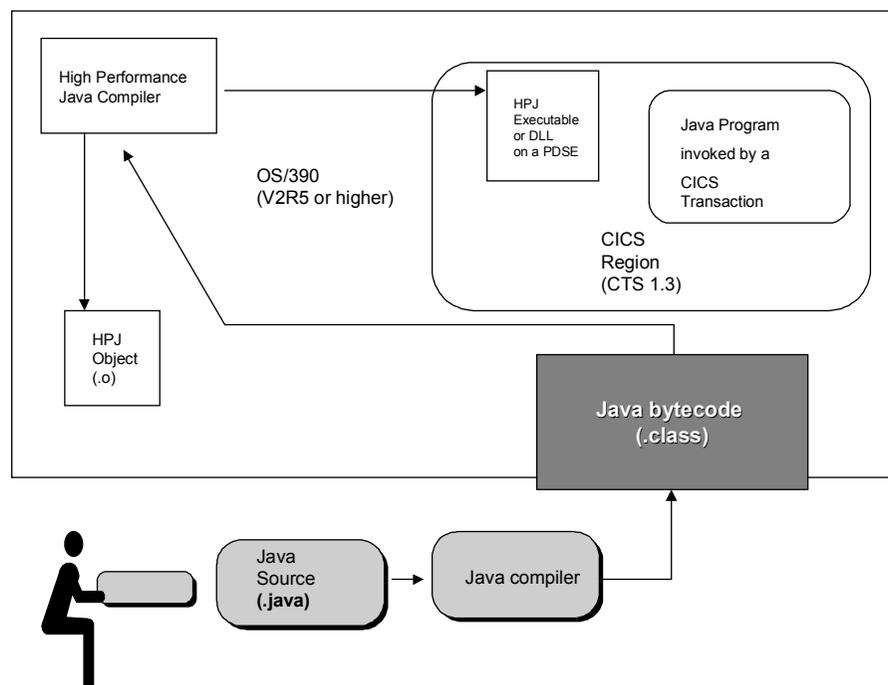
STROBE also supports measurement of Java applications built with IBM's high performance Java compiler running in a CICS Transaction Server 1.3 and higher environment on OS/390. To create these Java applications, the Java compiler creates bytecode from Java source code.

The Java bytecode is further bound and compiled by IBM's high performance Java compiler to produce a High Performance Java program object, which can be an executable or a DLL. The High Performance Java executable or DLL executes in a CICS TS 1.3 region. With STROBE you can measure the Java program when it is executed by a CICS transaction.

In the OS/390 environment, IBM's High Performance Java compiler statically compiles Java bytecode directly into native (object) code. Unlike just-in-time (JIT) compilers used by Java Virtual Machines, this OS/390 static compilation occurs only once, before execution time. Since it compiles only once, the amount of optimization the compiler can perform is greater than that with the JIT compiler.

The high performance Java compiler can use resource-intensive optimization techniques such as dataflow analysis and interprocedural optimization of IBM's common XL and OS/390 C/C++ compiler family. This compiler fully binds the code in an executable or dynamic link library (DLL) in a partitioned data set extended (PDSE) that can be run under the CICS Transaction Server for the OS/390. This enables you to write your enterprise's CICS applications in Java and optimize code performance on the server.

**Figure 1-3.** Overview of Java Execution in a CICS TS 1.3 Environment



---

## Where to Find More Information

You can find detailed instructions for submitting and managing measurement requests and creating Performance Profiles in the STROBE/ISPF online Tutorial and in Chapter 2 of the *STROBE MVS User's Guide* or in Chapter 2 of the *STROBE MVS User's Guide with Advanced Session Management*. For more information on interpreting reports in the Performance Profile, see *STROBE MVS Concepts and Facilities*.

## Chapter 2.

# Using the STROBE Java Feature

Using STROBE to measure a Java application differs depending on it is running in a JVM environment or the HPJ applications. For JVM, this chapter explains the following types of support for measurement with STROBE Java Feature:

- JVM Java program running in running an OMVS batch environment
- JVM Java program executed by CICS transactions running in a CICS environment
- JVM Java program calling DB2 databases using the JDBC and JSQL interfaces.

For HPJ, the chapter explains using STROBE to measure a HPJ Java program executed by CICS transactions in a CICS environment.

---

## Requirements for the STROBE Java Feature

Depending on whether you are using STROBE to measure HPJ Java applications or Java applications written using Java JVM, there are certain requirements that must be met before you can measure and produce a STROBE Performance Profile.

### JVM Java Requirements

When you measuring applications running under the JVM environment, you must have:

- Java Virtual Machine Version 1.3.1
- STROBE UNIX System Services Feature
- To measure Java programs in a CICS environment, you must have the STROBE CICS Feature installed.
- To obtain data reflecting SQL and Java in the JVM Performance Profile reports, you must have the STROBE DB2 Feature installed.

### HPJ Java Requirements

To measure HPJ applications, the Feature requires that you build the Java program with IBM's High Performance Java compiler packaged with IBM's VisualAge for Java for OS/390. When you compile Java source code for CICS, you include the Java class files that are output from the Java compiler in the CLASSPATH.

To measure a Java application with the STROBE Java Feature, you must also do the following:

- Compile with the binder option EDIT=YES (the default) of the High Performance Java compiler to ensure that control section information is complete.

Do not specify the COMPRESS option of the High Performance Java compiler.

---

## Measuring the Application

You use the same procedures to submit STROBE requests regardless of whether you are measuring a JVM or HPJ application. This section explains submitting a request to measure a job as it executes or to measure a job that has not yet executed.

### Measuring an Active Job

To measure an active job or online region, select Option 1 (ADD ACTIVE) from the STROBE OPTIONS menu. Complete the following information on the STROBE - ADD ACTIVE panel:

1. Specify the target system in the SYSTEM field or clear this field to select from a list of systems.
2. Specify the target job name in the JOBNAME field or clear this field to select from a list of active jobs.
3. Configure the measurement session by specifying
  - how long you want to measure the job in the SESSION DURATION field
  - how many performance samples you want STROBE to take in the TARGET SAMPLE SIZE field
  - the TSO user ID for STROBE to notify when the request is complete in the TSO USER ID TO NOTIFY field
4. Specify the sample data set information in the SAMPLE DATA SET INFORMATION fields.
5. Specify any additional measurement options by entering “Y” in the corresponding field and then pressing **Enter**. You can
  - specify special data collectors
  - specify additional module mapping facilities
  - add session management parameters
  - request a retention time frame
  - specify Java targeting information

STROBE displays the selected panels. When you press **Enter** on the last panel, STROBE submits the measurement request. See the *STROBE MVS User's Guide* or the *STROBE MVS User's Guide with Advanced Session Management* for more information on these measurement options.

6. Press **Enter** to submit the request.

### Measuring a Job That is not yet Executing

To add a measurement request for a job that is not yet executing, select Option 2 (ADD QUEUED) from the STROBE OPTIONS menu. Complete the following information in the STROBE - ADD QUEUED REQUEST panel:

1. Specify the target job name in the JOBNAME field.
2. Specify the target job step in the PROGRAM or STEP fields.
3. Specify the target system in the SYSTEM field or clear this field to select from a list of systems.
4. Configure the measurement by specifying
  - the estimated run time in the SESSION DURATION field
  - how many performance samples you want STROBE to take in the TARGET SAMPLE SIZE field
  - the TSO user ID for STROBE to notify when the request is complete in the TSO USER ID TO NOTIFY field

5. Specify the sample data set information in the SAMPLE DATA SET INFORMATION fields.
6. Specify any additional measurement options by entering “Y” in the corresponding field and then pressing **Enter**. You can
  - specify special data collectors
  - specify additional module mapping facilities
  - add session management parameters
  - request a retention time frame
  - specify Java targeting information

STROBE displays the selected panels. When you press **Enter** on the last panel, STROBE submits the measurement request. See the *STROBE MVS User's Guide* or the *STROBE MVS User's Guide with Advanced Session Management* for more information on these measurement options.

7. Press **Enter** to submit the request.

**Note:** In the JVM application environment, an OMVS batch job step can create multiple address spaces. You may need to examine these address spaces to determine exactly which one requires STROBE measurement. Or you can issue multiple ADD QUEUED measurement requests for the job step.

## Configuring Java Data Collection

To measure a Java application running in either a HPJ or JVM environment, there is only one requirement: to make sure the Java data collector is invoked. STROBE automatically invokes the Java data collector . If necessary, you can specify “Y” next to the JAVA field on the STROBE - DATA COLLECTORS panel to invoke the data collector. To suppress the collection of Java data, enter “N” in this field as shown in Figure 2-1 on page 2-3.

If you are measuring Java running under JVM, you can specify certain targeting criteria than enables you to select portions of the Java code that you want to measure. The next section “Java Targeting and Reporting” explains how to make these specifications.

**Note:** When measuring Java applications executing in a CICS environment or accessing DB2, do not specify “N” for the CICS or the DB2 data collector options on the STROBE - DATA COLLECTORS panel.

**Figure 2-1.** STROBE - DATA COLLECTORS Panel

```

----- STROBE - DATA COLLECTORS -----
COMMAND ==>

OVERRIDE DATA COLLECTOR DEFAULTS FOR JOBNAME: WPAJEA
DATA COLLECTORS: (Y or N; Y adds to and N removes from your system defaults)
  ADABAS      ==>   ADA3GL      ==>   C           ==>
  CICS        ==>   COBOL       ==>   CSP          ==>
  DB2         ==>   IDMS        ==>   IDMS BATCH DML ==>
  IEF         ==>   IMS         ==>   JAVA        ==> Y
  MQSERIES    ==>   NATURAL     ==>   PL/I         ==>
  SVC         ==>   VSAM        ==>

CICS Options:
  Collect Region Data ==>          OR   Produce Performance Supplement ==>
  Detail Transaction (TRAN or TR*): ==>          Collect Terminal Activity ==>
=>      =>      =>      =>      =>

CAPTURE Options: (Y or N; default is Y)
  DB2      ==>   IMS      ==>
  JVM targeting data (Y/N ==> Y

MQ Common User Module   ==>          Always use as default (Y/N) ==>

OTHER DATA COLLECTORS:

```

## Java Targeting and Reporting

By default, the STROBE Java Feature will attempt to attribute CPU or wait time to the main user method of the Java programs seen executing in an address space by examining the Java call stack that is in place when STROBE takes samples. As Java methods execute, memory in the Java call stack is allocated to them in the form of “stack frames”. As methods are invoked and then return, these frames are pushed and popped on and off of the call stack. At the time of a STROBE sample, the Java call stack state reflects the chain of methods from the first method invoked, usually the *main* method, to the method currently executing.

STROBE searches the call stack from the current method back to the first method in the stack. When it finds *main*, it stops and collects data about it and the method immediately invoked by *main*.

If you enter Java targeting information, STROBE performs an examination of the calling stack based on the provided targeting input. As it traverses the call stack, STROBE stops searching when it finds a call stack entry with a method name matching any of the targeting criteria or reaches *main*.

If multiple Java targeting criteria is specified, STROBE stops at the first call stack entry with a method name exactly matching the criteria or is called *main*. If STROBE finds neither *main* or a targeted method, it continues until it reaches the bottom of the call stack and attributes activity to the associated method.

For example, if the *main* method of class HelloWorld was invoked and STROBE sampled when the call stack was in the following state:

```
/my/test/HelloWorld.main(String[])
->/my/test/HelloWorld.SendGreeting(String[])
   ->/my/test/HelloWorld.formatData(String[])
       ->/java/lang/String.trim()
```

By default, the CPU or wait time for that sample would be associated with */my/test/HelloWorld.main(String[])*. STROBE would also note that *main* had invoked */my/test/HelloWorld.SendGreeting(String[])*.

## Using Java Targeting

In the above example, you could have targeted packages, classes and methods that you wanted STROBE to measure. If you had targeted */my/test/HelloWorld.formatData*, then STROBE would have attributed the CPU or wait time to that method. And it would report that the method had invoked */java/lang/String.trim()*. If you specified targeting data of */my/test/HelloWorld.GetGreeting*, STROBE would not have found the method in the call stack. Instead, it would stop at *main* and associate the CPU time with */my/test/HelloWorld.main(String[])*.

In certain situations, targeting is more useful than others. For example, if you measure a Java application and examine the Java CPU Usage by Called Method report, you would see that a certain called method appears to be involved in a logic path that consumed a large percentage of CPU time. You could target that method and measure the application again. The next set of Performance Profile reports would contain information about the methods that had been called by the targeted method. By continuing this approach to targeting, you can drill down as far as you want into a Java application’s activity.

**Note:** Performance Profiles for applications or classes without *main* methods (such as EJBs) may contain information about system methods only. In these cases, you should target a method of interest, such as an EJB business method or any other user-written method or class. The Performance Profile will then provide more detail about user-written methods instead of system methods.

## Targeting JVM Java Code for Measurement

For the JVM environment, there are a set of STROBE ISPF panels you can use to specify measurement target information. Entering “Y” in the “JVM targeting data” field show in Figure 2-1 causes STROBE to display the STROBE - JAVA TARGETING ISPF panel shown in Figure 2-2 on page 2-5.

**Figure 2-2.** STROBE - JAVA TARGETING Panel

```

----- STROBE - JAVA TARGETING -----
COMMAND ==>

List java targets ==> N (Y or N; default N)
OR
Enter Packages, Classes or Methods() to target below
Line commands      E -Extended Edit
Targeting data types P -Package C -Class M -Method

Lc Type

- - -----
- - -----
- - -----
- - -----

```

### Specifying Target Information

You can target Java code in a JVM environment where you want STROBE to focus either by specifying it yourself or by selecting from a STROBE-generated list that shows the contents of the Java packages and classes you want to measure. The section “Using the STROBE Java Target Selector” on page 2-6 explains how to select measurement targets off of this list.

If you want to enter the Java targeting information yourself make sure that the specifications you make are as precise as possible. For example, some methods are part of multiple classes. To enable STROBE to measure exactly what you want in the Java program, you should provide as much detail in your targeting information as possible.

**Note:** Unless you have already determined what specific methods you want to measure, you should not use targeting the first time you measure a Java application running in the JVM environment. By first creating a STROBE Performance Profile that shows all of the Java activity for the measurement request, you can select packages, classes and methods that showed performance problems. Then you can target what you have identified as containing improvement opportunities and create another Performance Profile.

Follow these instructions, to target Java code for STROBE measurement.

1. In the **Type** field, identify the type of Java code that you will specify on the following lines:
  - **P** indicates you are specifying a Java Package
  - **C** indicates you are specifying a Java Class or a Package and Class combination



- EAR file name
  - JAR file name
  - WAR file name
  - ZIP file name
  - Package name
  - Class File name (This can be the name of the ZIP file in which the Java package is contained.) You can also enter the exact name of a class file. If you leave this entry blank, but have specified a location on the first set of lines, then a list of all packages in that location and its subdirectories is returned.
3. Press ENTER, and the Java targeting selector will return a list of packages. If you enter just the class name , the list only provides a class name and the methods it contains.

**Note:** Using the selector to obtain Java targets from locations where a large number of classes or files exist can return an excessive number of targets. To minimize the amount of data returned, the targeting information should be as specific as possible.

**Figure 2-4.** STROBE - JAVA TARGETING LIST Panel

```

----- STROBE - JAVA TARGETING LIST-----
COMMAND ==>

Enter location of java targets (blank if in classpath)

_____

Enter file, package or class name to target

  com.compuware.strobe

Enter 'S' to select Package, Class, or Method to target:
Enter '+' against Class to list Methods:

-----
Package: com.compuware.strobe
- Class():Class
-   method: read()
-   method: toString()
-   method: write()
- Class(-):Directory
  S  method: find()
-   method: list()
  S  Class(+):File

```

The targeting list shows what exactly is contained in the Java packages classes and methods as specified by the targeting information you provided.

4. Entering "S" next to a class or method name indicates this is where you want STROBE to focus the measurement. If you enter a plus "+" sign, next to a class name, then all the methods contained within that class are listed. You can then select the method by entering "S" next to its name. .
5. Once you press ENTER after all targets have been selected, their names will be displayed on the STROBE - JAVA TARGETING LIST ISPF panel. Press ENTER again and the measurement request is submitted by STROBE.

## Creating a Performance Profile

The STROBE Performance Profile contains base STROBE reports and reports specific to the STROBE Java Feature. STROBE automatically generates the Java reports if the Feature has collected Java-specific data.

Depending on whether you are measuring HPJ applications or programs written using JVM, you will get a different set of reports. See Chapter 3 for examples and descriptions of the HPJ reports and Chapter 4 for example and descriptions of the JVM reports..

## Controlling the Level of Detail in the Reports

By specifying certain STROBE options, you can either compress or expand the level of detail in the Performance Profile reports.

You can control the size of the reports by specifying certain parameters when you create the Performance Profile. For example, on the reports listed below, you can specify a minimum percentage of activity that must occur for a procedure or cylinder to appear as an individual line on the report. If the minimum percentage is not attained, contiguous procedures or cylinders are condensed on a single line in the report. The lower the number you specify, the greater the level of detail in the reports.

The following reports have minimum percentage options:

- Program Usage by Procedure
- DASD Usage by Cylinder
- Transaction Usage by Control Section
- the Attribution report

You can break the reports into manageable sections by specifying the report resolution — the number of bytes considered to be a codeblock for detailed reporting. The lower the number you specify, the greater the level of detail in the reports. For more information, see *STROBE MVS Concepts and Facilities*.

## Tailoring the STROBE HPJ Java Feature Reports

To tailor the reporting of HPJ Java information, you may specify the following parameters in the OTHER PARAMETERS field of the STROBE - DETAIL FOR A PERFORMANCE PROFILE panel:

- The JAVARPT=*nn.nn* (default 00.01%) This parameter enables you to condense activity reported on the CPU Usage by Java Method report. The value *nn.nn* indicates the minimum percentage of solo and total CPU time that STROBE reports. STROBE suppresses solo and total CPU time if the solo CPU time is less than this percentage.
- NOJAVRPT This parameter enables you to suppress the Java Class Summary and the CPU Usage by Java Method reports.

To suppress the Java Attribution of CPU Execution Time report enter “Y” in the Java field in the ATTRIBUTION Reports section of the Tailor Reports panel as shown in Figure 2-5.

Figure 2-5. STROBE - TAILOR REPORTS Panel

```

----- STROBE - TAILOR REPORTS -----
COMMAND ==>

WAIT TIME BY MODULE -- Show location of wait ==> (Specify Y)
----- Report ----- Compress below OR Suppress (Specify Y)
PROGRAM USAGE BY PROCEDURE ==> % ==>
DASD USAGE BY CYLINDER ==> 02.0 % ==>
TRANSACTION USAGE BY CONTROL SECTION ==> % ==>
CICS TRANSACTION PROFILE ==> sec ==>
CICS REGION LEVEL ==> sec ==>
MQSERIES CALLS ==> % ==>
ATTRIBUTION Reports ==> % ==>
  Suppress reports for:
    C ==> CICS ==> COBOL ==>
    CSP ==> DB2 ==> DL/I ==>
    IDMS ==> IEF ==> JAVA ==> Y
    MQSERIES ==> PL/I ==> SVC ==>
PROGRAM SECTION USAGE SUMMARY Display inactive ==>
TIME and RESOURCE DEMAND DISTRIBUTION
  Combine tasks ==> Display all tasks ==> Display all DDs ==>

CICS TRANSACTION PROFILE FILTERS => => => => =>
  Suppress non-CICS TRANSACTION REPORTS ==> (Specify Y)
USE DATE AND TIME FORMAT FROM PARMLIB ==>

```



## Chapter 3.

# Analyzing a HPJ Java Performance Profile

Once your measurement is complete and you have created a Performance Profile, your next step is to examine the Profile and identify performance improvement opportunities. Once you have identified the performance improvement opportunities and incorporated the indicated changes into the Java application, you can then measure the application again with STROBE and produce a Performance Profile to verify the effects of your changes.

This chapter explains the specialized reports produced by the STROBE Java Feature for HPJ Java applications and one standard STROBE report that is tailored to report Java information. The Feature produces two reports that detail execution of CPU time in Java classes and methods. STROBE produces all the standard STROBE Performance Profile reports, including an Attribution of CPU Execution Time report that identifies Java classes and methods that invoked system services. A detailed description of the additional standard STROBE reports is provided in *STROBE MVS Concepts and Facilities*.

---

## Finding Significant Activity in the Java Application

To find CPU activity within the Java application, analyze the Java Class Summary, the CPU Usage by Java Method, and the Attribution of CPU Execution Time reports. The Java Class Summary and the CPU Usage by Java Method reports are produced only by the STROBE Java Feature and show CPU consumption by Java classes and, within classes, by Java methods. The Attribution of CPU Execution Time report is a standard STROBE report tailored to report Java-specific information.

### Java Class Summary Report

The Java Class Summary report (Figure 3-1 on page 3-2) provides an overview of the CPU usage in the classes within each module that STROBE measured. This section describes each of the fields on this report.

#### MODULE NAME

The name of the module which is responsible for CPU usage.

**Note:** For all module or section names that exceed eight characters, STROBE generates a *token*, which is an eight-byte identifier. The token comprises the first four characters of the module or section name followed by a hyphen (-) and then the last three characters of the name. Refer to the Token - Longname Cross Reference report to reconcile all tokens with their long names. For more information on this report see Chapter 3 of *STROBE MVS Concepts and Facilities*.

#### CLASS NAME

The name of the Java class within the module identified in the MODULE NAME field for which STROBE is detailing CPU usage.

**Note:** For all class names that exceed eight characters, STROBE generates a *token*, which is an eight-byte identifier. The token comprises the first four characters of the module or section name followed by a hyphen (-) and then the last three characters of the name. Refer to the Token - Longname Cross Reference report to

reconcile all tokens with their long names. For more information on this report see Chapter 3 of *STROBE MVS Concepts and Facilities*.

### CLASS SIZE

The decimal size of the class in bytes.

### % CPU TIME

The total percent of CPU time used in class code plus system services activity attributed to class code.

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O or CPU activity.

### CPU TIME HISTOGRAM

The histogram shows the distribution of CPU usage within each class. Solo CPU time is indicated with the symbol “\*\*”. The remaining CPU time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

Figure 3-1. Java Class Summary Report

** JAVA CLASS SUMMARY **									
MODULE NAME	CLASS NAME	CLASS SIZE	% CPU TIME		CPU TIME HISTOGRAM			MARGIN OF ERROR:	
			SOLO	TOTAL	.00	.50	1.00	1.50	2.00
JCICSMED	JCICSMED	23648	.13	.13	.**				
JCICSMED	TOTALS		.13	.13					
JAVA CLASS	TOTALS		.13	.13					

The CPU Usage by Java Method report (Figure 3-2 on page 3-3) shows execution and attribution activity for a Java module, class, or method. In the header lines, the report reflects the lowest level at which the Feature detected activity. In the detail lines the report indicates the offset within the lowest level—the module, class, or method—at which the Feature detected activity.

In the header lines this report lists:

### MODULE

The name of the module for which STROBE is detailing CPU usage.

### CLASS

The name of the class within the module identified in MODULE for which STROBE is detailing CPU usage.

### METHOD

The name of the method within the class identified in CLASS for which STROBE is detailing CPU usage. For all method names that exceed eight characters, STROBE generates a *token*, which is an eight-byte identifier. The token comprises the first four characters of the module or section name followed by a hyphen (-) and then the last three characters of the name. Refer to the Token - Longname Cross Reference report to

reconcile all tokens with their long names. For more information on this report see Chapter 3 of *STROBE MVS Concepts and Facilities*.

Each detail line on the report lists:

**OFFSET**

The location by offset within the module, class, or method at which STROBE detected activity. The lowest level in the header line indicates whether this is an offset within a module, class, or method.

**STATEMENT NUMBER**

The line number of the Java program that corresponds to the offset within the module, class, or method at which STROBE detected activity.

**% CPU TIME**

The percent of CPU time used by the module, class, or method that exceeds the value specified with the JAVARPT parameter (**default .01%**).

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O activity.

**CPU TIME HISTOGRAM**

The histogram shows the distribution of CPU usage within each class. Solo CPU time is indicated with the symbol “\*”. The remaining CPU time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

**Figure 3-2. CPU Usage by Java Method Report**

```

** CPU USAGE BY JAVA METHOD **

MODULE - JCICSMED
CLASS - JCICSMED
METHOD - main-er) - main(com.ibm.cics.server.CommAreaHolder)

  OFFSET          STATEMENT          % CPU TIME   CPU TIME HISTOGRAM   MARGIN OF ERROR:  1.02%
                   NUMBER                SOLO   TOTAL      .00   .50   1.00   1.50   2.00
0000066E         00000052          .01    .01    .    .
00000D5C         00000065          .12    .12    .**  .
-----
CLASS  JCICSMED TOTALS          .13    .13
-----
MODULE JCICSMED TOTALS          .13    .13
    
```

**Attribution of CPU Execution Time Report**

The Attribution of CPU Execution Time report (Figure 3-3 on page 3-4) shows the Java method within the Java class that invoked a system service. Each Attribution report detail line identifies a location from which the service routine was directly or indirectly invoked. The location is defined by module, class, method, and return address.

If the method did not directly invoke the service routine, the routine first called by the invoking location is defined under “via” by module and control section with a brief description of its function.

The report lists:

**XACTION**

The name of the transaction that invoked the service routine, if there was one.

**MODULE**

The name of the module that invoked the service routine.

**CLASS**

The name of the class within the module identified in MODULE that invoked the service routine.

**METHOD**

The name of the method within the class identified in CLASS that invoked the service routine.

**RETURN**

The location, in hexadecimal, where the application invoked the service routine.

**MODULE (under VIA)**

The name of the module that invoked the service routine if it was not directly invoked.

**SECTION (under VIA)**

The name of the control section that invoked the service routine if it was not directly invoked.

**FUNCTION**

A brief description of the function of the control section that invoked the service routine if it was not directly invoked.

**CPU TIME %**

The percentage of all CPU execution time used by programs executed within the address space that was spent in the invoked system service routine on behalf of the invoking routine.

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O activity.

**Figure 3-3. Attribution of CPU Execution Time Report**

** ATTRIBUTION OF CPU EXECUTION TIME **										
.LELIBS	CEEPLPKA	LE/370 VECTOR CSECT			-----VIA-----			CPU TIME %		
XACTION	MODULE	SECTION	RETURN	LINE	PROCECURE NAME	MODULE	SECTION	FUNCTION	SOLO	TOTAL
MED	DFJCZDTC		003C34			CEEPLPKA		LE/370 VECTOR CSECT	.14	.14
MED	DFJCZDTC		003C7C			CEEPLPKA		LE/370 VECTOR CSECT	.04	.04
XACTION	MODULE	CLASS	METHOD	RETURN		MODULE	SECTION	FUNCTION	SOLO	TOTAL
MED	JCICSMED		main-er)	00066e		JCICSMED	__hp-rap		2.37	2.37
									-----	-----
									2.55	2.55

## Chapter 4.

# Analyzing a JVM Java Performance Profile

If you are measuring a JVM Java application with STROBE, the Performance Profile contains a different set of reports than if you were measuring a HPJ Java program. The following lists the reports that are produced for a JVM-written application:

- Java CPU Usage and Wait Time by Called Method Summary reports present execution information at the method level. Processing and wait are related back to the user targeted method or the initial method run.
- Java CPU Usage and Wait Time by Called Method reports show the package, class and methods directly invoked by the user targeted or initial method. Additionally, invoked DB2 statements are shown.
- Java CPU Usage and Wait Time by Executing Method reports show the method or service routine actually processing at the time STROBE samples. (These reports distinguish between interpreted and Just-In-Time “JITted” compiled code.)
- Java Targeting - This report shows the names of the search arguments used by the Java targeting function.
- Java Environment - This report shows CLASSPATH information for each JVM profile.

**Note:** If you have the STROBE CICS Feature installed, the CICS API and Non-API transaction profile reports show Java methods invoked under CICS.

This chapter explains the specialized reports produced by the STROBE Java Feature and one standard STROBE report that is tailored to report Java information. The Feature produces two reports that detail execution of CPU time in Java classes and methods. STROBE produces all the standard STROBE Performance Profile reports, including an Attribution of CPU Execution Time report that identifies Java classes and methods that invoked system services. A detailed description of the additional standard STROBE reports is provided in *STROBE MVS Concepts and Facilities*.

Compuware’s iSTROBE product enables you to view and analyze STROBE Performance Profile data on a workstation using a standard Web browser. If you use iSTROBE to analyze a JVM Java application, you have access to much more detailed information about called methods, as described in “The iSTROBE Java Activity by Called Method Report” on page 4-7.

---

## Measurement Session Data Report

The Measurement Session Data Report enables you to validate the Performance Profile and helps you focus your effort to improve application performance. If you have the STROBE Java Feature installed, the Java data collector is active, and you targeted Java code for measurement, this report will indicate that a Java targeting report will be generated.

Figure 4-1. Measurement Session Data Report

```

** MEASUREMENT SESSION DATA **

----- JOB ENVIRONMENT -----
PROGRAM MEASURED - DFHSIP
JOB NAME - CICSPROD
JOB NUMBER - STC 1427
STEP NAME - CICSPR02

----- MEASUREMENT PARAMETERS -----
ESTIMATED SESSION TIME - 45 MIN
TARGET SAMPLE SIZE - 10,000
REQUEST NUMBER - 26
FINAL SESSION ACTION (A)- QUIT

----- MEASUREMENT STATISTICS -----
CPS TIME PERCENT - 13.90
WAIT TIME PERCENT - 86.10
RUN MARGIN OF ERROR PERCENT - .98
CPU MARGIN OF ERROR PERCENT - .13

----- REPORT PARAMETERS -----
REPORT RESOLUTION - 64 BYTES
SORTSIZE - 999,999
LINES/PAGE - 60

DATE OF SESSION - 09/23/02
TIME OF SESSION - 10:45:16
CONDITION CODE - C-0000

SYSTEM - z/OS 01.04.00
DFSMS - 1.3.0
SUBSYSTEM - CICS TS 2.2
JVM 1.3.1
UNIX SERVICES
CPU MODEL - 3090-400E
SMF/SYSTEM ID - ASYS
LPAR - C222
64-BIT ARCHITECTURE ENABLED

DASD= 1.0% DASDGAP=5

REGION SIZE BELOW 16M - 7,104K
REGION SIZE ABOVE - 32,768K
DATE FORMAT - MM/DD/YY
TIME FORMAT (24 HOURS) - HH:MM:SS

PTF LEVEL - 3.0.000/000

TOTAL SAMPLES TAKEN - 10,000
TOTAL SAMPLES PROCESSED - 10,000
INITIAL SAMPLING RATE - 3.70/SEC
FINAL SAMPLING RATE - 3.70/SEC

SESSION TIME - 49 MIN 0.78 SEC
CPU TIME - 6 MIN 53.95 SEC
WAIT TIME - 42 MIN 42.35 SEC
STRETCH TIME - 0 MIN 0.00 SEC

SRB TIME - 0 MIN 0.24 SEC
SERVICE UNITS - 344
PAGES IN - 0 OUT- 16
PAGING RATE - 0.01/SEC
EXCPS - 113,803 38.70/SEC

SAMPLE DATA SET - SA.CICSPROD.S001D001

```

**SUBSYSTEM**

This field will show the Java release you are using.

**JAVA TARGETING DATA**

If this field contains data, targeting information was specified for the measurement request. shows all the targeting data for Java measurements you provided as described in Chapter 2. The Java Targeting Information report shows a complete list of targeting arguments.

**Java Targeting Information Report**

The Java Targeting Information report shown in Figure 4-2 contains the parameters you specified to target Java code.

Figure 4-2. Java Targeting Information Report

```

** JAVA TARGETING INFORMATION **

TARGETING SEARCH ARGUMENT(S) : METHOD: com/timbow/services/business/TargetsService/validateTarget
                             PACKAGE:com/timbow/tracing
                             CLASS: com/timbow/services/business/Calculator

```

**TARGETING SEARCH ARGUMENTS**

This field lists all of the Java targeting information you provided to STROBE to select Java code for measurement.

## Java Environment Report

The Java Environment report shown in Figure 4-3 contains CLASSPATH information for JVM profiles. For each profile, the report lists:

**Figure 4-3.** Java Environment Report

```

** JAVA ENVIRONMENT **

PROFILE : DFHJVMPR
CLASSPATH - /usr/lpp/cicsts/cts220/samples/dfjcics
            /u/sb/wpama/classes:
            /u/sb/wpajma:
            /u/sb/wpanjk1/java:
            /u/sb/wpanjk0:
            /usr/lpp/db2/db2710/classes/db2j2classes.zip:
            /u/sb/pbtsdc0/java/sqlj/CUP222_CICS/CupCICS.jar

```

### PROFILE

This field provides the name of the JVM profile that was measured by STROBE. If you measuring in a CICS Transaction Server environment, more than one profile may appear in the report with its associated CLASSPATH. If STROBE measures a batch region, this field is blank.

### CLASSPATH

This field shows the CLASSPATH environment variable which is used by the JVM to find user-defined class libraries.

---

## Java CPU Reports

This section provides examples and descriptions of the Performance Profile reports that contain information about CPU usage for the targeted methods. The Java “called method” report attributes execution activity back to the ultimate invoker or application method. The Java “executing” method report only contains information about the method STROBE found executing at the time it took a sample.

### Java CPU Usage by Called Method Summary Report

The Called Method Summary report shown Figure 4-4 on page 4-4 provides information at the method level. The method name is composed of (if present, the package name), the class name, the method name and the argument list. Processing time is related back to the application method (the *main* method).

STROBE attempts to attribute CPU or wait time to the main user method of the Java programs seen executing in an address space by examining the Java call stack that is in place when STROBE takes samples. It searches the call stack from the current method back to the first method in the stack. When it finds *main*, it stops and collects data about it and the method immediately invoked by *main*.

As STROBE examines the Java call stack, it may find either or both the class constructor and initialization methods. The constructor method is the initialization method that is called automatically when an object is created. It is identified by the syntax “<init>”.

The class initialization method is called by the JVM to initialize the created object’s static variables. It is identified by the syntax “<clinit>”. Both the <init> and <clinit> methods will appear in Performance Profile reports when STROBE is able to collect measurement data for them.

In this example, the greatest CPU usage (45.65%) was attributed to method `main(com/ibm/cics/server/CommAreaHolder)`.

**Figure 4-4.** Java CPU Usage by Called Method Summary Report

```

** JAVA CPU USAGE BY CALLED METHOD SUMMARY **

METHOD SIGNATURE                                % CPU TIME SOLO  CPU TIME TOTAL  HISTOGRAM  MARGIN OF ERROR: 1.20%
                                                    46.00
com/ibm/cics/server/Wrapper.<clinit>()          11.09  .11.09  .*****
com/ibm/cics/server/Wrapper.main(java/lang/string)  .69  .69  .
com.ibm.jvm.Trace.initializeTrace()            .16  .16  .
com/timbow/appl/cta/CTAInitialSelections.main(com/ibm/ci 45.63  45.65  .*****
cs/server/CommAreaHolder)
java/lang/ref/Finalizer.<clinit>()              .01  .01  .
java/lang/Character.<clinit>()                  .30  .30  .
java/lang/Class.<clinit>()                      .07  .07  .
java/lang/ClassLoader.getSystemClassLoader()   1.13  1.13  .
java/lang/ClassLoader.loadClass(java/lang/String) 11.21  11.22  .*****
java/lang/Compiler.<clinit>()                  .12  .12  .
java/lang/System.initializeSystemClass()        .40  .40  .
java/lang/Thread.init(java/lang/ThreadGroup,java/lang/Ru
nable,java/lang/String)
java/lang/ThreadGroup.<init>(java/langThreadGroup,java/
lang/String)
java/util/jar/JarFile.ibmJVMtidyUp()           .01  .01  .
sun/io/CharacterEncoding.<clinit>()             .06  .06  .
sun/misc/signal.<clinit>()                     .01  .01  .
COM/ibm/db0s390/sqlj/jdbc/DB2SQLJDriver.ibmJVMtidyUp() .03  .03  .
-----
JAVA METHOD TOTALS                               71.07  71.07

```

#### METHOD/SIGNATURE

Name of the method and argument list associated with the method. If a method has more than one argument lists, each instance of the method name and each argument list is considered a separate method.

#### % CPU TIME

The total percent of CPU time used on behalf of this method plus system services activity attributed to class code.

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O or CPU activity.

#### CPU TIME HISTOGRAM

The histogram shows the distribution of CPU usage within each class. Solo CPU time is indicated with the symbol “\*”. The remaining CPU time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

#### MARGIN OF ERROR

The margin of error for CPU time percentages. This value also appears in the CPU MARGIN OF ERROR PCT field on the Measurement Session Data report.

#### JAVA METHOD TOTALS

Total amount of CPU usage for the reported methods.

## Java CPU Usage by Called Method Report

This report shows the package, class and methods directly invoked by the user targeted or initial application method. For each class, the total CPU time for all of its methods is

reported. Figure 4-5 on page 4-5 shows an example of the report. The methods listed below were all directly invoked by *main(com/ibm/cics/server/CommAreaHolder)*. The CPU time associated with the called method is the percentage of CPU time use that can be attributed to the logic path beginning with *CommAreaHolder()* calling the method associated with the CPU time.

**Figure 4-5.** Java CPU Usage by Called Method Report

```

** JAVA CPU USAGE BY CALLED METHOD **

PACKAGE - com/timbow/appl/cta
CLASS - CTAInitialSelectins
METHOD - main(com/ibm/cics/server/CommAreaHolder)

PACKAGE/CLASS
METHOD          % CPU TIME  CPU TIME HISTOGRAM  MARGIN OF ERROR:  2.61%
                SOLO   TOTAL   .00   .50   1.00   1.50   2.00
  INVOKED SQL

com/timbow/appl/ca/CTAConcert
toTarget(java/lang/Object)          5.77   5.77   .*****
CLASS CATConvert                    TOTAL   5.77   5.77

com/timbow/appl/cta/CTAInitialSelections
getData()                            .03   .03   .
CLASS CTAInitialSelections          TOTAL   .03   .03

com/timbow/business/procedure/matching/CTAInterpreter
<clinit>()                          .45   .45   .
CLASS CTAInterpreter                TOTAL   .45   .45

com/timbow/services/business/TargetsService
findTargets(com/timbow/business/entities/TargetsFile,
com/timbow/business/procedure/atching/ITargetRes
ultsInterpreter)                    35.85  35.87  .*****
  SQL OPEN  CMPWR312  0-01   .22   .22   .
  SQL OPEN  CMPWR312  0-02   .19   .19   .
CLASS TargetService                  TOTAL   36.27  36.27

com/timbow/service/persistence/DataAccessService
writePerformanceStatistics()         .07   .07   .
CLASS DataAccessService              TOTAL   .07   .07

com/timbow/services/persistence/Loader
build/TargetsFile(long,
com/tibow/services/persistence/persistenceMetadat
a)                                   2.51   2.51   .**
  SQL SELECT CMPWR342  0-00   .01   .01   .
  SQL SELECT CMPWR342  0-01   .01   .01   .
  SQL SELECT CMPWR342  0-02   .01   .01   .
CLASS Loader                          TOTAL   2.56   2.56

java/io/PrintStream
println(java/lang/String)            .03   .03   .
CLASS PrintStream                    TOTAL   .03   .03

java/lang/ClassLoader
loadClass(java/lang/String)         .27   .27   .
CLASS ClassLoader                    TOTAL   .27   .27

java/lang/Throwable
printStackTrace()                   .01   .01   .
CLASS Throwable                       TOTAL   .01   .01

METHOD com/timbow/appl/cta/CTAInitialSelections.  TOTAL   45.56  45.57
main(com/ibm/cics/server/CommAreaHolder)

```

**PACKAGE**  
**CLASS**  
**METHOD**

The name of the package and class and method that is the ultimate invoker of the rest of the classes and methods listed in the report.

**PACKAGE/CLASS  
METHOD  
INVOKED SQL**

The name of the method and the class containing it that was invoked by the the initial user application method or the user-targeted method. The method name(s) is indented two characters past the class name. Any SQL statement that was invoked by the called method, either directly or via lower levels of called methods is listed for the method and class and is indented two characters past the method name.

After the SQL statement name, the DBRM name is listed. Then the report will show a pair of numbers separated by a dash. The first number is the SQL statement number if one can be determined by STROBE. (Generally, the SQL statement number cannot be determined in a Java environment.) The second number is a STROBE generated number. You can use the number generated by STROBE and the DBRM name to be able to cross-reference the STROBE DB2 Feature Performance Profile reports to obtain additional SQL report information.

**% CPU TIME**

The total percent of CPU time used in class code plus system services activity attributed to class code.

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O or CPU activity.

**CPU TIME HISTOGRAM**

The histogram shows the distribution of CPU usage within each class. Solo CPU time is indicated with the symbol “\*”. The remaining CPU time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

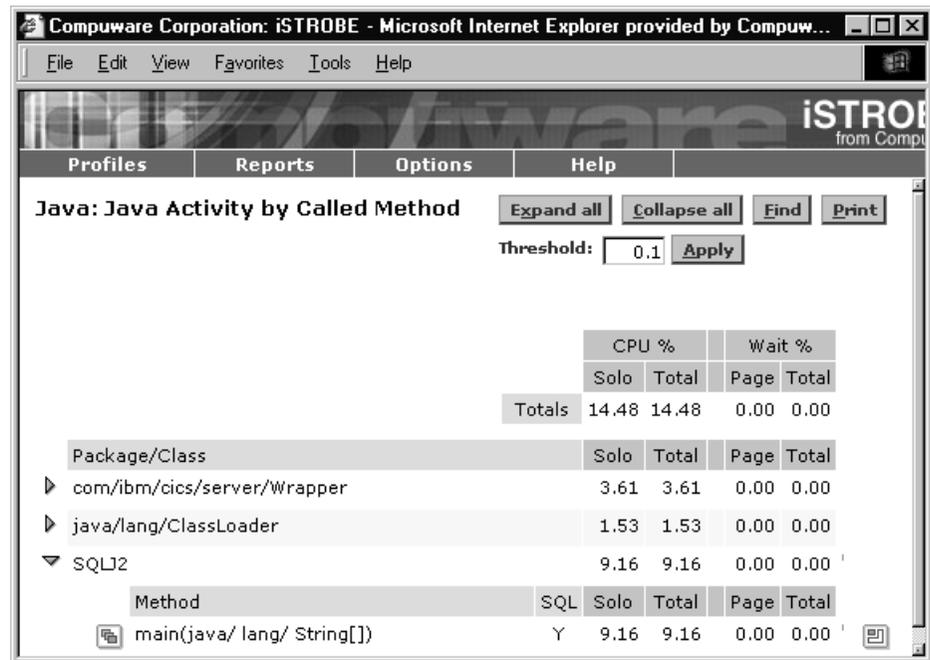
**CLASS TOTAL**

Total amount of CPU usage for the reported classes.

## The iSTROBE Java Activity by Called Method Report

Users of iSTROBE have a greatly enhanced ability to analyze Java JVM measurement data. The STROBE printed Performance Profile shows only two layers of a call stack, while iSTROBE provides access to as many as eight. To view stack data in iSTROBE, start by selecting the Java Activity by Called Method report (Figure 4-6 on page 7).

Figure 4-6. iSTROBE Java Activity by Called Method Report

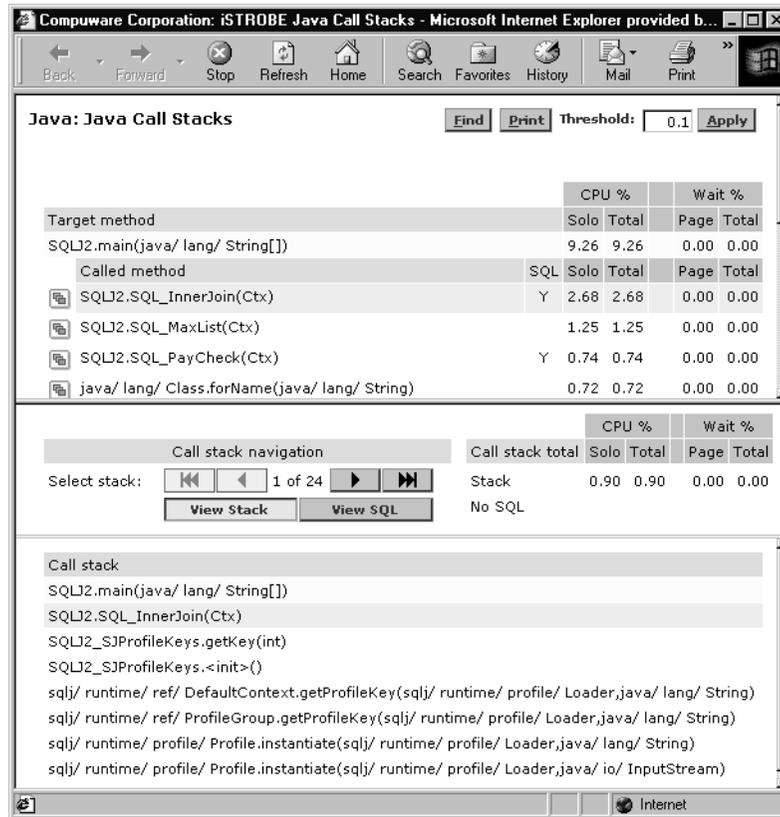


Like the STROBE Called Method reports, this iSTROBE report shows the package, class and methods directly invoked by the user-targeted or initial application method. However, in iSTROBE:

- Both CPU and wait activity are conveniently aggregated into one report.
- The **Threshold** field enables you to modify the minimum percentage of CPU or wait that a method must have used if it is to appear on the report.
- You can choose exactly which Package/Class you wish to examine in more detail, rather than viewing the called methods for all of them.

Click the stacks icon  next to a method to see the Java Call Stacks report, shown in Figure 4-7.

Figure 4-7. iSTROBE Java Call Stacks Report



For the method you select, the Java Call Stacks report shows all the methods that it called directly. For each of these called methods, you can view the contents of its associated call stacks, along with any SQL statements attributed to them.

iSTROBE online help provides a detailed description of the data and functions provided by these reports. To view it, from the iSTROBE Java Activity by Called Method, select **Help, Help for current page**.

## Java CPU by Executing Method Report

This report shows CPU execution in a Java method. A Java method may be executing as machine instructions or as bytecode. If an asterisk "\*" appears next to a method name in the report, it indicates the the JVM was executing and interpreting the method's bytecode. Figure 4-8 shows an example of the report.

Figure 4-8. Java CPU By Executing Method Report

```

** JAVA CPU BY EXECUTING METHOD **

NOTE: * DENOTES INTERPRETED METHOD

PACKAGE/CLASS          % CPU TIME  CPU TIME HISTOGRAM  MARGIN OF ERROR:  1.20%
METHOD                SOLO   TOTAL .00   .50   1.00   1.50   2.00

com/ibm/jvmTrace
*initializeTrace()    .01   .01   .
-----
CLASS Trace          TOTAL    .01   .01

com/ibm/mqservices/Trace
*trace(int,java/lang/String,java/lang/String)  .01   .01   .
-----
CLASS Trace          TOTAL    .01   .01

com/ibm/ras/RASMaskChangeGenerator
*init()              .01   .01   .
-----
CLASS RASMaskChangeGenerator  TOTAL    .01   .01

com/timbow/appl/cta/sntranslation/SNResponder
*initPI()            .01   .01   .
-----
CLASS SNResponder      TOTAL    .01   .01

com/timbow/appl/cta/snstrnslation/
SDRTargetsFileTranslator
*createBytes(com/timbow/services/translator/IMessage)  .01   .01   .
-----
CLASS SDRTargetsFileTranslator  TOTAL    .01   .01

com/timbow/appl/cta/sntranslation/SDRUtil
*setByteArrayDate(byte,int,int,int,java/lang/String)  .01   .01   .
setByteArrayDate(byte,int,int,int,java/lang/String)  .01   .01   .
-----
CLASS SDRUtil          TOTAL    .03   .03

```

#### PACKAGE/CLASS METHOD

The name of the package/class and method that was actively processing at the time STROBE sampled.

#### % CPU TIME

The total percent of CPU time used in class code.

- SOLO shows activity without any concurrent I/O or CPU activity being performed under control of programs executing within the address space.
- TOTAL shows activity with or without any concurrent I/O or CPU activity.

#### CPU TIME HISTOGRAM

The histogram shows the distribution of CPU usage within each class. Solo CPU time is indicated with the symbol “\*”. The remaining CPU time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

#### CLASS TOTAL

Total amount of CPU usage for the reported classes.

## Java Wait Time Reports

This section provides examples of the Performance Profile reports containing wait time information for JVM Java applications.

**Note:** The CICS Transaction Profile reports will contain Java execution measurement, but wait time is not recorded for a CICS address space.

### Java Wait Time by Called Method Summary Report

This report shows information at the method level. The method name is composed of (if present, the package name), the class name, the method name and the argument list. Wait time is related back to the application method (the *main* method).

**Figure 4-9.** Java Wait Time by Called Method Summary Report

** JAVA WAIT TIME BY CALLED METHOD SUMMARY **						
METHOD SIGNATURE	% RUN TIME PAGE	RUN TIME TOTAL	HISTOGRAM	MARGIN OF ERROR:	.72%	
com/ibm/mq/MqThread.run()	.00	35.64	.+++++	18.00	27.50	36.00
com/timbow/appl/CollectStatisticsByRequest/CollectStatistic.main(java/lang/string)	.00	.18	.			
JAVA METHOD TOTALS	-----	-----				
	3.21	39.04				

#### METHOD/SIGNATURE

Name of the method and argument list associated with the method. If a method has more than one argument lists, each instance of the method name and each argument list is considered a separate method.

#### RUN TIME PERCENT

The percentage of time during the measurement session that the address space was in the wait state. There are two measures of run time:

- PAGE shows wait time that results from retrieving a page from the page data set. A high value in this column indicates that there is not enough physical memory assigned to the address space. If you noticed a high paging rate on the Measurement Session Data report, the this report enables you to see which method was experiencing delay because of paging.
- TOTAL measures all causes of wait time, including page retrieval, programmed I/O operations, and timer requests.

#### RUN TIME HISTOGRAM

The histogram shows the distribution of run time within each class. Solo run time is indicated with the symbol “\*”. The remaining run time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

#### MARGIN OF ERROR

The margin of error for wait time percentages. This value also appears in the RUN MARGIN OF ERROR PCT field on the Measurement Session Data report.

#### JAVA METHOD TOTALS

Total amount of wait time for the reported methods.

## Java Wait Time by Called Method Report

This report shows the package, class and methods directly invoked by the user targeted or initial application method. For each class, the total wait time for all of its methods is reported. Figure 4-10 on page 4-11 shows an example of the report. In this example, the initial method is *main(java/lang/string)*. The methods listed below were all directly invoked by this method. The wait time associated with the called method is the percentage of wait time use that can be attributed to the logic path beginning with *java/lang/string* calling the method associated with the wait time.

**Note:** For an explanation of the advantages of using the iSTROBE version of this report, see “The iSTROBE Java Activity by Called Method Report” on page 4-7.

**Figure 4-10.** Java Wait Time by Called Method Report

```

** JAVA WAIT TIME BY CALLED METHOD **

PACKAGE - com/timbow/appl/CollectStatisticsbyRequest/
CLASS   - CollectStatistics
METHOD  - main(java/lang/string)

PACKAGE/CLASS          % RUN TIME  RUN TIME HISTOGRAM  MARGIN OF ERROR:  2.61%
METHOD                PAGE    TOTAL  .00      .50      1.00      1.50      2.00
INVOKED SQL

com/tibow/appl/SelectStatus
  processRequestStats()      .00    .00
  SQL SELEC CMPWR252  0-07  .00    .18
CLASS SelectStatus          TOTAL  .00    .18

```

### PACKAGE CLASS METHOD

The name of the package and class and method that is the ultimate invoker of the rest of the classes and methods listed in the report.

### PACKAGE/CLASS METHOD INVOKED SQL

The name of the method and the class containing it that was invoked by the the initial user application method or the user-targeted method. The method name(s) is indented two characters past the class name. Any SQL statement that was invoked by the called method, either directly or via lower levels of called methods is listed for the method and class and is indented two characters past the method name.

### RUN TIME PERCENT

The percentage of time during the measurement session that the address space was in the wait state. There are two measures of run time:

- PAGE shows wait time that results from retrieving a page from the page data set. A high value in this column indicates that there is not enough physical memory assigned to the address space. If you noticed a high paging rate on the Measurement Session Data report, this report enables you to see which method was experiencing delay because of paging.
- TOTAL measures all causes of wait time, including page retrieval, programmed I/O operations, and timer requests.

**RUN TIME HISTOGRAM**

The histogram shows the distribution of run time within each class. Solo run time is indicated with the symbol “\*”. The remaining run time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

**MARGIN OF ERROR**

The margin of error for wait time percentages. This value also appears in the RUN MARGIN OF ERROR PCT field on the Measurement Session Data report.

**JAVA METHOD TOTALS**

Total amount of wait time for the reported methods.

**CLASS TOTAL**

Total amount of wait time for the reported classes.

**Java Wait Time by Method Report**

This report shows wait time in a Java method. Figure 4-11 on page 4-12 shows an example of the report.

**Figure 4-11.** Java Wait Time by Method Report

```

** JAVA WAIT TIME BY METHOD **

NOTE: * DENOTES INTERPRETED METHOD

PACKAGE/CLASS                                % RUN TIME  RUN TIME HISTOGRAM  MARGIN OF ERROR: 2.61%
METHOD                                         WAIT   TOTAL  .00    1.00    3.00    3.00    4.00

com.ibm.or.gapache.xml.utiles.synthetic.Buildme
  declareInnerClass(String);                  .14    .14    .**
  *toSource(PrintStream, int);                 .05    .05    .*
  toSource(PrintStream, int);                 1.45   1.45   .*****
  writeObjects(String, BuildMe, Object, PrintStream,
    MaterialCompositionDetector);             .14    .14
CLASS BuildeMe                                TOTAL  1.78   1.78

com.ibm.org.apache.xml.utils.synthetic.
MaterialCompositionDetector
  *findCompositionOfMaterial(PrintStream, int); 1.35   1.35   .*****
  findCompositionOfMaterial(PrintStream, int); 1.15   1.15   .*****
CLASS MaterialCompositionDetector            TOTAL  2.50   2.50

java.io.PrintStream
  print(String);                              .70    .70   .*****
  println(String);                            .21    .21   .****
CLASS PrintStream                            TOTAL  .91    .91

TOTAL                                         -----
                                         5.19   5.19

```

**PACKAGE/CLASS  
METHOD**

The name of the package/class and method that was the cause of wait at the time STROBE sampled.

**RUN TIME PERCENT**

The percentage of time during the measurement session that the address space was in the wait state. There are two measures of run time:

- PAGE shows wait time that results from retrieving a page from the page data set. A high value in this column indicates that there is not enough physical memory assigned to the address space. If you noticed a high paging rate on the Measurement Session Data report, this report enables you to see which method was experiencing delay because of paging.
- TOTAL measures all causes of wait time, including page retrieval, programmed I/O operations, and timer requests.

**RUN TIME HISTOGRAM**

The histogram shows the distribution of run time within each class. Solo run time is indicated with the symbol “\*”. The remaining run time is indicated with the symbol “+”. Spikes, or lengthy lines in the histogram, highlight classes with a high proportion of activity.

**MARGIN OF ERROR**

The margin of error for wait time percentages. This value also appears in the RUN MARGIN OF ERROR PCT field on the Measurement Session Data report.

**CLASS TOTAL**

Total amount of wait time for the reported classes.

**Program Section Usage Summary Report**

The Program Section Usage Summary report shown in Figure 4-12 provides the distribution of CPU time used by each active control section of each Java module in the measured address space. Examine this report to determine what programs or systems services are consuming the most CPU resources. You can then look up those programs in the Program Usage by Procedure report to determine which procedures cause the consumption.

**Figure 4-12.** Program Section Usage Summary Report

** PROGRAM SECTION USAGE SUMMARY **									
MODULE NAME	SECTION NAME	16M <,>	SECT SIZE	FUNCTION	CPU TIME PERCENT		CPU TIME HISTOGRAM		MARGIN OF ERROR: 2.61%
					SOLO	TOTAL	.00	13.50	27.00 40.50 54.00
.SYSTEM	.ISG			GRS	7.78	7.78			.*****
.SYSTEM	.JAVA			JAVA	5.19	5.19			.***
.SYSTEM	.LELIB			LE/370 LIBRARY SUBROUTNE	3.40	3.40			.**
.SYSTEM	.NUCLEUS			MVS NUCLEUS	1.77	1.77			.*
.SYSTEM	.SMS			SYSTEM MANAGER STORAGE	16.84	16.84			.*****
.SYSTEM	.USS			UNIX SYSTEM SERVICES	52.58	52.58			.*****
					-----	-----			
.SYSTEM	TOTALS			SYSTEM SERVICES	87.56	87.56			

**MODULE NAME**

The name of the module or pseudo-module for which STROBE is detailing CPU usage. For module names that exceed eight characters, see the note in the SECTION NAME paragraph. A module name of JAVAJIT indicates activity was observed in Java methods that had been compiled to machine code. The “Java CPU by Executing Method Report” on page 4-8 and the “Java Wait Time by Method Report” on page 4-12 includes JAVAJIT activity.

**SECTION NAME**

The name of the control section or the pseudo-section within the module or pseudo-module identified in MODULE NAME for which STROBE is detailing CPU usage. A section name may not appear if the activity in the section is less than the percentage specified in the MODULE MAPPING BASELINE (default 2%) or if STROBE was unable to map the module. STROBE is unable to index a section that is not mapped. A section name of .JAVA indicates activity in either HPJ Java modules or JVM system modules.

For all module or section names that exceed eight characters, STROBE generates a *token*, which is an eight-byte identifier. The token comprises the first four characters of the module or section name followed by a hyphen (-) and then the last three characters of the name. Refer to the Token - Longname Cross Reference report to reconcile all tokens with their long names.

**Note:** STROBE reports "?LONG*nnn*" for the section names that exceed eight characters if initialization of the long names data space fails. The suffix "*nnn*" is a sequence number beginning with "001". The suffix is reset to "001" for each module. If initialization of the long names data space fails, STROBE also does not produce the Token - Longname Cross Reference report.

**16M < , >**

Indicates whether the RMODE of a module is 24 bit (<) or 31 bit (>). If the module has multiple control sections, this indication appears next to TOTALS in the SECTION NAME column. If the module is a Generalized Object File Format (GOFF) split-RMODE module an "S" appears in this column. If a GOFF module contains multiple control sections, the RMODE indicator appears for each control section.

**SECT SIZE**

The size of the control section in bytes, expressed in decimal notation. The report does not show section size for pseudo-sections, such as .IOCS and .SVC, that group related system services together and summarize those services.

**FUNCTION**

A short description of the function of the control section or pseudo-section or of the module or pseudo-module. Function descriptors appear for all pseudo-sections and pseudo-modules and, if your STROBE system programmer has supplied them, for other control sections and modules as well.

**CPU TIME PERCENT**

The percentage of all CPU time executing on behalf of the control section. There are two measures of CPU time:

- SOLO shows activity without any concurrent I/O activity for the target program or subsystem.
- TOTAL shows activity with or without concurrent I/O activity.

If a module contains more than one control section, the report also shows subtotals by control section. The Program Usage by Procedure report treats each control section in the Program Section Usage Summary report in greater detail.

**CPU TIME HISTOGRAM**

Displays the intensity of CPU usage within each control section. Solo CPU time is indicated by the symbol "\*". The remaining CPU time is indicated by the symbol "+". Spikes, or lengthy lines in the histogram, highlight control sections with a high proportion of activity.

**MARGIN OF ERROR**

The margin of error for the CPU time percentages, also reported on the Measurement Session Data report. This margin of error, which appears in the header line, applies only to the number of samples in which STROBE found the CPU to be active.

## Program Usage by Procedure Report

The Program Usage by Procedure report shown in Figure 4-13 details the time the CPU spent executing Java code within each area of each control section of each module of the program or subsystem. The report is ordered alphabetically by module and, within module, by control section. Examine this report when the Program Section Usage Summary or the Most Intensively Executed Procedures reports show a concentration of CPU use. This report normally appears in two formats: a report formatted for system modules and a report formatted for user-written modules.

**Figure 4-13.** Program Usage by Procedure Report

** PROGRAM USAGE BY PROCEDURE **										
MODULE NAME	.SYSTEM SECTION NAME	SYSTEM SERVICES FUNCTION	INTERVAL LENGTH	.JAVA SOLO	JAVA % CPU TIME TOTAL	CPU TIME HISTOGRAM	MARGIN OF ERROR:	1.53%		
						.00 .50	1.00	1.50	2.00	
.JAVA	com.-dME	JAVA		1.78	1.78	*****				
.JAVA	com.-tor	JAVA		2.50	2.50	*****				
.JAVA	com.-eam	JAVA		.91	.91	*****				
	.JAVA	TOTALS		5.19	5.19					

This report shows CPU usage for all system service routines under the pseudo-module .SYSTEM. A report for a system module begins with a header line that shows the pseudo-module (.SYSTEM) and the pseudo-section, with a description of its function.

Detail lines show:

- **MODULE NAME**, the true module name or an SVC identified by number. The module name JAVAJIT indicates activity was observed in Java methods that had been compiled to machine code. The “Java CPU by Executing Method Report” on page 4-8 and the “Java Wait Time by Method Report” on page 4-12 details JAVAJIT activity.
- **SECTION NAME**, the control section name (if STROBE obtained one during sampling). If the section name is a token, refer to the Token - Longname Cross Reference report to resolve the token name.
- **FUNCTION**, the function descriptor of the control section (if available) or the function descriptor of the module. Function descriptors do not appear for STROBE Features not installed at your site.

## Attribution Reports

The STROBE Java Feature produces two attribution reports.

### Attribution of CPU Execution Time Report

When relevant, attribution of CPU activity seen in system routines or Java methods that have been JIT compiled is attributed to the initial application method or the user-targeted method. See *STROBE MVS Concepts and Facilities* for more information about this report.

**Figure 4-14.** Attribution of CPU Execution Time Report

```

** ATTRIBUTION OF CPU EXECUTION TIME **
.USSS  BXINPVT      VECT TBL PRIV AREA MOD
-----
XACTION PACKAGE/CLASS/METHOD  WAS INVOKED BY----- VIA----- -CPU TIME %-
                                     PACKAGE/CLASS/METHOD  SOLO  TOTAL
TRANA   java-e()                com.ng)                .14   .14
                                     -----
                                     .14   .14
    
```

### Attribution of CPU Wait Time Report

When it relevant, attribution of wait time seen in system routines or Java methods that have been JIT compiled is attributed to the initial application method or the user-targeted method. See *STROBE MVS Concepts and Facilities* for more information about this report.

**Figure 4-15.** Attribution of CPU Wait Time Report

```

** ATTRIBUTION OF CPU WAIT TIME **
.USSS  BXINPVT      VECT TBL PRIV AREA MOD
-----
XACTION PACKAGE/CLASS/METHOD  WAS INVOKED BY----- VIA----- WAIT TIME %-
                                     PACKAGE/CLASS/METHOD  PAGE  TOTAL
TRANA   java-e()                com.ng)                .00   .01
                                     -----
                                     .00   .01
    
```

# Index

## Special Characters

.SYSTEM  
in Program Usage by Procedure report, 4-15  
?LONGnnn, defined, 4-14

## A

active jobs, measuring, 2-2  
adding  
  active measurement requests, 2-2  
additional measurement options, specifying, 2-2  
Attribution of CPU Execution Time report, 3-3  
  using to identify CPU time, 1-2  
attribution reports, showing system services, 3-3  
attribution, described, 1-2

## B

batch processing applications, measuring, 2-2  
benefits of using STROBE  
  overview, 1-1

## C

CICS Transaction Server 1.3, 1-5  
configuring measurement sessions, 2-2  
control sections  
  displaying inactive, 4-13  
controlling the level of detail in reports, 2-8  
CPU  
  time  
    in Program Section Usage Summary report, 4-13  
CPU TIME HISTOGRAM  
  in Program Section Usage Summary report, 4-14  
CPU TIME PERCENT column  
  in Program Section Usage Summary report, 4-14  
CPU Usage by Java Method report, 3-2

## D

data collectors, specifying, 2-2

## F

field, described, 1-3

## FUNCTION

column  
  in Program Section Usage Summary report, 4-14  
  in Program Usage by Procedure report, 4-15  
function descriptors  
  of control sections or pseudo-control sections, 4-14

## H

High Performance Java compiler, 1-5

## I

iSTROBE  
  advantages of, 4-7  
  Java Activity by Called Method report, 4-7  
  Java Call Stacks report, 4-8

## J

Java Class Summary report, 3-1  
Java CPU by Executing Method report, example, 4-8  
Java CPU Usage by Called Method report, example, 4-4  
Java CPU Usage by Called Method Summary report, example, 4-3  
Java environment report, example, 4-3  
Java programming language, described, 1-3  
Java targeting information report, example, 4-2  
Java Virtual Machine, 1-5  
Java Wait Time by Called Method Summary report, example, 4-10  
JAVARPT, 2-8  
job name, specifying, 2-2  
JVM environment targeting, 2-5

## M

MARGIN OF ERROR field  
  in Program Section Usage Summary report, 4-14  
  in Program Usage by Procedure report, 4-4, 4-10, 4-12–4-13  
measurement requests  
  described, 1-2  
Measurement Session Data report, example, 4-1  
measurement sessions  
  configuring, 2-2  
  described, 1-2  
measurement task, 4-1  
measuring application performance, benefits of, 1-1  
measuring the application  
  active jobs, 2-2  
  adding session management parameters, 2-2  
  batch processing applications, 2-2  
  online regions, 2-2  
  requesting retention, 2-2

- specifying additional measurement options, 2-2
- specifying data collectors, 2-2
- specifying module mapping facilities, 2-2
- method, described, 1-3
- minimum percentage options, reports with, 2-8
- module mapping facilities, specifying, 2-2
- MODULE NAME column
  - in Program Section Usage Summary report, 4-13
  - in Program Usage by Procedure report, 4-15

## N

NOJAVARPT, 2-8

## O

object, described, 1-3  
 online applications, measuring, 2-2

## P

PAGE wait column, in Wait Time by Module report, 4-10–4-11, 4-13

parameters

- JAVARPT, 2-8
- NOJAVARPT, 2-8

Performance Profile  
 described, 4-1

Program Section Usage Summary report  
 columns

- 16M, 4-14
- CPU TIME PERCENT, 4-14
- FUNCTION, 4-14
- MODULE NAME, 4-13
- SECT SIZE, 4-14
- SECTION NAME, 4-14
- SOLO, 4-14
- TOTAL, 4-14

CPU TIME HISTOGRAM, 4-14

MARGIN OF ERROR, 4-14

Program Usage by Procedure report, 4-4, 4-10, 4-12–4-13, 4-15

columns

- FUNCTION, 4-15
- MODULE NAME, 4-15
- SECTION NAME, 4-15

fields

- MARGIN OF ERROR, 4-4, 4-10, 4-12–4-13

function of, 4-15

relation to Program Section Usage Summary report, 4-14

## R

reports

- Attribution of CPU Execution Time, 1-2, 3-3
- controlling level of detail in, 2-8
- controlling size of, 2-8
- CPU Usage by Java Method, 3-2
- Java Class Summary, 3-1

- showing system services invoked by the application, 3-3
- specifying resolution, 2-8
- with minimum percentage options, 2-8
- resolution, specifying in reports, 2-8
- retention, requesting, 2-2
- RUN TIME PERCENT column
  - in Wait Time by Module report, 4-10–4-11, 4-13

## S

sample data sets

- described, 1-2
- specifying, 2-2

SECT SIZE column, in Program Section Usage Summary report, 4-14

SECTION NAME column

- in Program Section Usage Summary report, 4-14
- in Program Usage by Procedure report, 4-15

session management parameters, adding, 2-2

SOLO column

- in Program Section Usage Summary report, 4-14

specifying

- additional measurement options, 2-2
- job names, 2-2
- report resolution, 2-8
- sample data sets, 2-2
- target systems, 2-2

STROBE

- benefits of using, 1-1
- defined, 1-1
- systems, specifying, 2-2

## T

targeting in JVM, 2-5

TOTAL  
 column

- in Program Section Usage Summary report, 4-14

wait column, in Wait Time by Module report, 4-10–4-11, 4-13

## U

using STROBE  
 terminology, 1-2

## W

Wait Time by Module report  
 columns

- PAGE wait, 4-10–4-11, 4-13
- RUN TIME PERCENT, 4-10–4-11, 4-13
- TOTAL wait, 4-10–4-11, 4-13