



# **WHITE PAPER**

## Application Performance Management

**A Practical Approach to Balancing Application Performance  
and J2EE Instrumentation Information**

## A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information

---

Abstract .....	3
Introduction .....	4
Intended audience and usage note .....	5
Measuring performance .....	6
Performance impacts explained.....	7
Increased startup times.....	7
Increased response times .....	7
Increased memory use .....	8
Increased CPU consumption .....	8
Managing performance impacts.....	10
Starting points .....	10
Instrumentation modes .....	10
Adaptive Instrumentation utility.....	12
The Instrumentation Explorer tool.....	13
Application server metrics.....	13
Ignore the junk .....	14
Instrumentation definitions .....	15
Custom instrumentation definitions .....	15
Conclusion .....	17

## **Abstract**

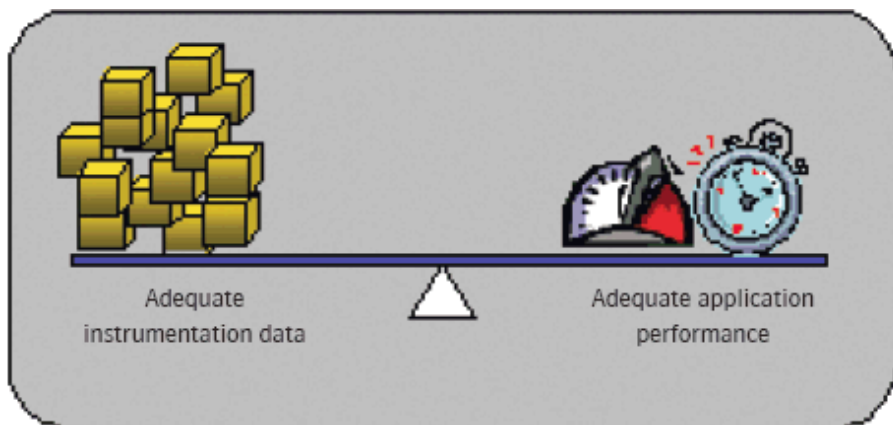
When using Java™ performance monitoring and analysis tools, the challenge is to gather the necessary and useful information without adversely affecting the performance of the application under test. Quite often, significant overhead is introduced in the target application due to the monitoring instrumentation. One workaround for this is to gather performance data and “take the hit” only in test, performance, or pre-production environments, not during production. This is an incomplete solution for a variety of reasons: it is inadequate in cases where performance issues only occur within the production environments; it cannot always be done, because there are situations where no adequate non-production environment exists; and finally, it is in production where performance really matters. For solutions such as Precise for J2EE that are designed for continual use within production, a balance between “enough data” and runtime overhead must be established and maintained over time.

The Precise for J2EE suite is not exempt from the issues outlined above. It is for this reason that tools and techniques have been developed to understand and minimize overhead. This brief examines the causes and impacts of overhead caused by instrumentation and the proper use of the tools within the Precise for J2EE suite to balance application performance and instrumentation value.

## **Introduction**

There is a phenomenon within science called the “observer effect.” The concept is simple: The act of observing will change the phenomenon being observed. For instance, to measure voltage in an electrical circuit, a voltmeter must be connected to the circuit—which changes the voltage that is being measured.

In much the same way, software performance-monitoring tools always have an impact on the performance of the software being measured. Ultimately it is the goal of the performance-monitoring tool to gather the maximum amount of data with the minimum impact to the application being monitored. The degree to which the tool can accomplish this is a key differentiator between vendors of performance tools. The software alone, though, is not solely responsible for attaining this goal. User decisions (and errors) can adversely affect the tool’s overall performance impact.



**Figure 1 - Striking the balance**

This paper will explain how we measure performance of an application, the basics of instrumentation, how instrumentation affects application performance, and how a balance between instrumentation and performance impacts can be achieved using Precise for J2EE.

## **Intended audience and usage note**

The general instrumentation and performance concepts presented in this paper should be educational for just about everyone, but some of the techniques described are intended for only the most experienced users of Precise for J2EE.

**PLEASE NOTE:** THE TOPICS COVERED HERE INCLUDE INSTRUCTIONS FOR ALTERING CORE INSTRUMENTATION FILES—ALTERATIONS THAT, IF MADE INCORRECTLY, MAY LEAD TO ISSUES THAT CAN ONLY BE REMEDIED BY RESTORATION OF THE DEFAULT CONFIGURATION. SUCH CHANGES SHOULD ONLY BE MADE BY EXPERIENCED USERS, FOLLOWING A FULL BACKUP OF THE INSTALL, AND WITH FULL UNDERSTANDING OF THE CONSEQUENCES.

## Measuring performance

Before discussing the impact of instrumentation upon application performance, it may be useful to define the measurements that are used to quantify application performance. The following vocabulary terms are metrics that will be used throughout this paper:

- **Application startup time**—the time elapsed between initial startup of an application and full readiness of the application for use; typically measured in seconds
- **Application response time**—the time required for any given application feature or transaction to complete (for example, "It takes 3 seconds to log in"); typically measured in seconds or milliseconds
- **Memory consumption**—the steady state value of the process memory for an application; typically measured in megabytes (MB)
- **CPU consumption**—the average steady state use of CPU by an application; typically measured as a percentage of the available CPU

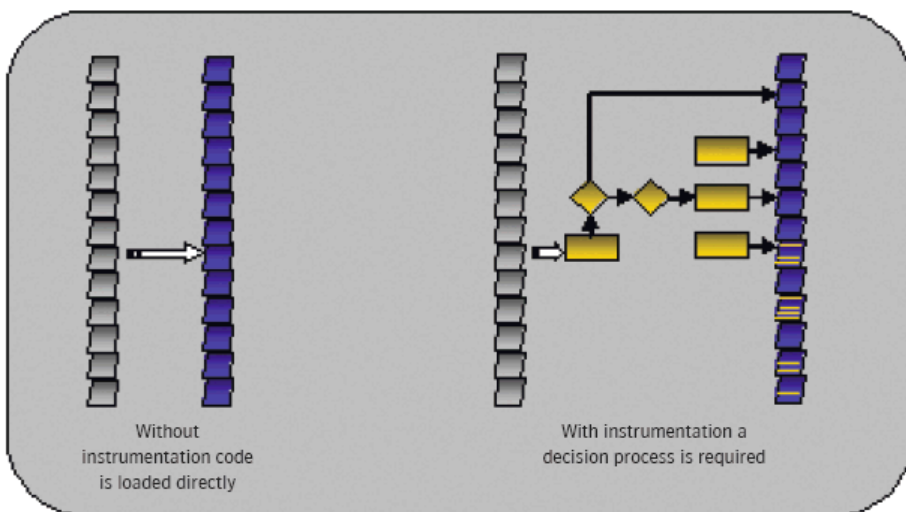


Figure 2 - Adding instrumentation takes time.

## Performance impacts explained

As we seek to minimize the impact of monitoring upon application performance, it is useful to understand clearly the ways a monitoring tool can affect the target application's performance metrics, as previously defined.

### Increased startup times

The process of adding valuable instrumentation points within an application is, by its nature, an intrusive one. Either the application code must be modified with instrumentation prior to execution or it must be otherwise bracketed with instrumentation. In any case, work must be done before the application is ready to be used. This work includes inspecting the code before loading it within the Java Virtual Machine (JVM); determining if the code should be instrumented and, if so, what type and quantity of instrumentation to add; and, finally, adding the instrumentation and loading the code. This process typically increases the application startup time and, in general, its scope of impact is a factor of the size of the application being instrumented and the quantity of instrumentation being added.

### Increased response times

This is perhaps the most obvious and visible way that application performance is impacted by monitoring tools. When the response times of your application lengthen, people tend to notice! In general, this happens when the instrumentation that is injected into the application cumulatively takes a noticeable amount of time to conduct its work. The use of the word "noticeable" here is important because changes in response time are very relative. Adding 100 milliseconds to an application call is a big deal when the un-instrumented call takes only 50 milliseconds to return (a 200% increase in response time!), but when the call being monitored takes 10 seconds to complete, 100 milliseconds is nothing.

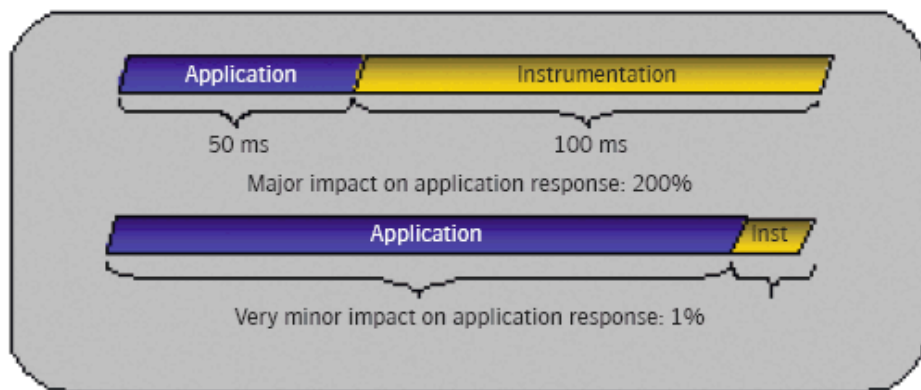


Figure 3 - Response time impacts are relative.

## A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information

The fact is that instrumentation code needs time, both CPU and temporal, to take measurements and perform the calculations necessary for data gathering. In general, this can be considered a linear impact.

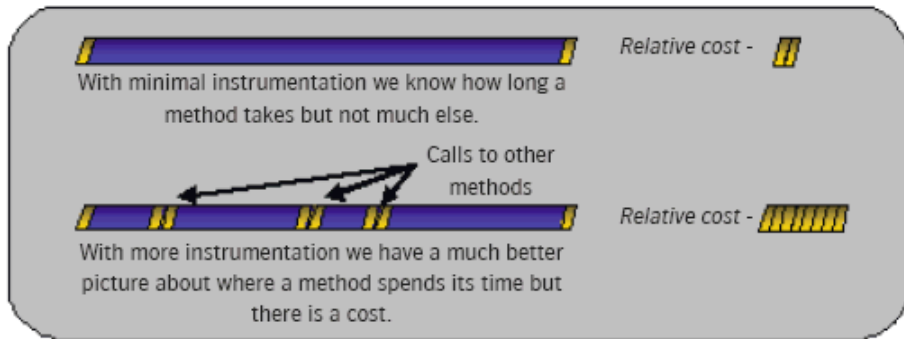


Figure 4 - Instrumentation data is not free.

### Increased memory use

One can imagine the amount of data required to record the location of every call from every method to every method within a given application. Now, compound this amount by saving the timing of each invocation, perhaps along with some meta-data associated with each call, and it is no wonder that adding instrumentation to an application requires more memory. In a well-designed monitoring tool, the memory impacts are understood and minimized—but again, you cannot expect to get meaningful and complete instrumentation data without seeing an impact on memory.

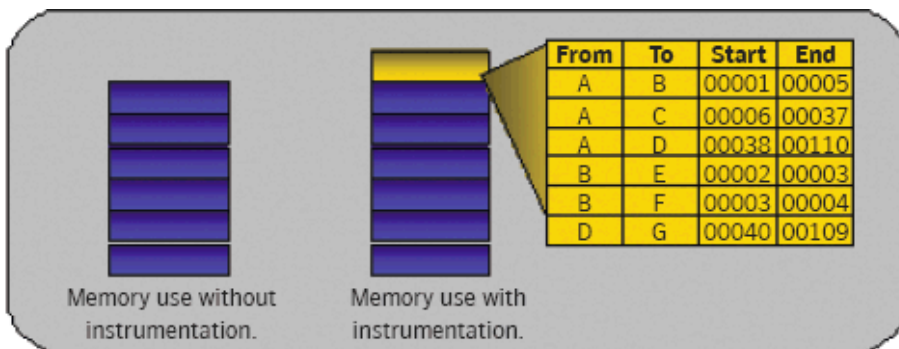


Figure 5 - Instrumentation data requires memory.

### Increased CPU consumption

At its core, a software monitoring tool is also just software. As such, it will require CPU cycles, as does any other piece of code running on a machine. This demand may be particularly noticeable during periods when the performance tool is summarizing or aggregating data or when the instrumentation data is being transferred to the reporting process.

## A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information

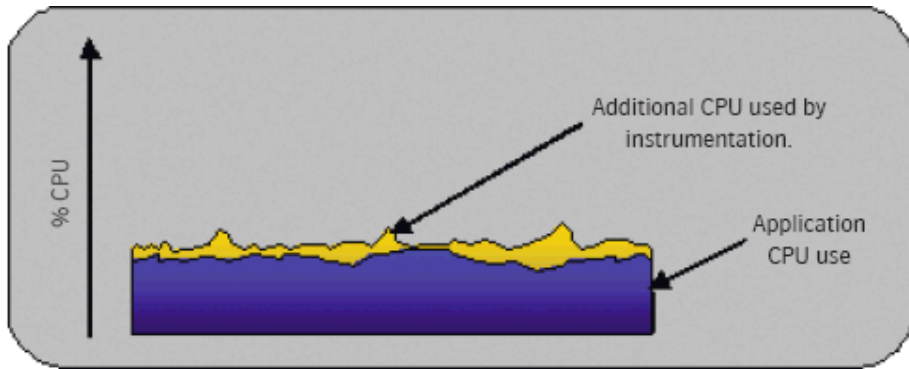


Figure 6 - Instrumentation uses CPU time.

The monitoring tool's need for CPU cycles can conflict with and potentially impact your application performance. This is particularly true if your server is under constant and significant load. If the monitoring tool pushes CPU demands past the available CPU, then both the monitoring tool and your application server will become CPU-bound and will experience CPU wait times.

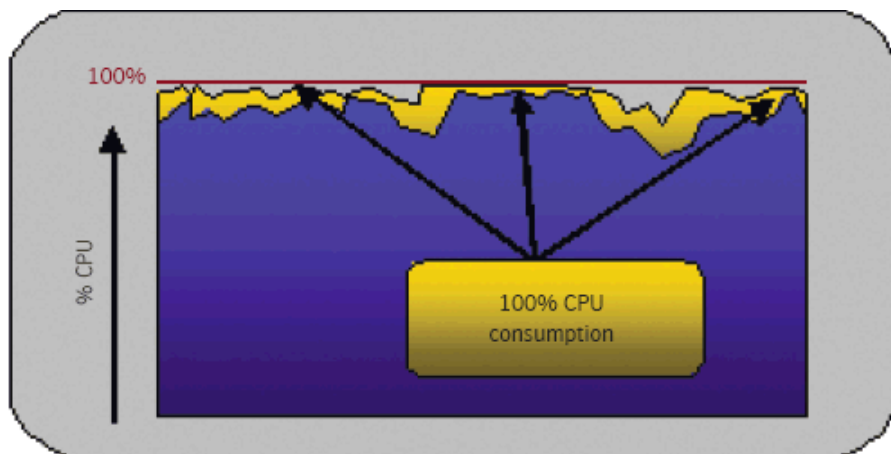


Figure 7 - Heavily loaded systems see higher relative impact of instrumentation.

## **Managing performance impacts**

As you can see, the act of adding performance-monitoring instrumentation to an application has an impact on that application's performance; in fact, this is a perfect example of the observer effect. Thus it is critical to balance the amount and type of added instrumentation with the quantity and quality of the performance data desired.

Luckily, using the Precise for J2EE tools and the techniques described in this paper, you should be able to understand the impacts of your decisions and manage instrumentation effectively to strike this delicate balance.

Put simply, the more information you want, the higher the price—in terms of impact on your application—you must be willing to pay!

## **Starting points**

When you first install Precise for J2EE and configure an application for instrumentation, the instrumentation is added according to a default set of rules. These rules have been defined by the development team to provide a good starting point for J2EE applications in general, but they are far from ideal for your application. The default configuration includes the following:

- The configuration captures many common instrumentation types, including simple HTTP, EJB, JDBC, and XML calls.
- A standard instrumentation mode provides visibility of standard J2EE interfaces as well as of non-standard Java/J2EE components. Initial overhead is relatively low, but dynamic instrumentation is limited.
- The instrumenter ignores a default set of classes/packages.

It is important to understand the starting instrumentation as you read through the following sections, which discuss the many ways to alter instrumentation and the corresponding impact on performance.

## **Instrumentation modes**

To provide users with maximum flexibility in balancing visibility vs. overhead, Precise for J2EE supports three modes of instrumentation: Basic, Standard, and Expert. The three modes differ in the following ways:

- Initial overhead (the overhead caused by the out-of-the-box monitoring probes)
- Users' flexibility to modify instrumentation using advanced instrumentation tools
- The cost of (dynamically) instrumenting additional components or methods



## A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information

Instrumentation modes compared			
Criteria	Basic	Standard	Expert
Startup time	Fastest	Fast	Slowest
Response time impact	Minimal	Minimal	Moderate
Memory increase	Minimal	Moderate	Moderate
Performance information	Minimal	Moderate	Maximum
Instrumentation flexibility	Minimal	Moderate	Maximum

### Instructions for advanced users

- The instrumentation mode for a given JVM is defined in the Precise for J2EE user interface (UI) and can be reached by the following navigation path:
  1. Launch the Precise for J2EE UI for the environment in question.
  2. Click on the Settings link.
  3. Click on the collector machine in question.
  4. Click on the JVMID of the JVM in question.
  5. Click on the Settings icon.
  6. A dialog window will open in which you can alter the instrumentation mode for this JVM.
  7. Click OK when complete.
- After altering the instrumentation mode, you must restart the JVM for the changes to take effect.

For more detailed information about the instrumentation modes, see the Precise for J2EE online help.

## Adaptive Instrumentation utility

Adaptive Instrumentation is perhaps the most powerful, yet simple tool to use in tuning your instrumentation. This utility takes a step-by-step wizard approach to collecting instrumentation data, analyzing instrumentation value/impact, and, finally, updating instrumentation configuration.

The basic idea of Adaptive Instrumentation is fairly simple: Take an instrumentation snapshot before and after a period of activity, and then analyze the differences. This analysis can yield valuable application information such as which methods are used the most (or least) and which methods yield the longest response times. This data can then be compared to a set of policies and an “overhead budget,” which is then used to auto-generate custom instrumentation.

Another benefit of this process is that the analysis can identify instrumentation that is having a significant runtime impact on the application, and disable it. Ideally, this process should be repeated several times, with a JVM restart after each repetition. Each iteration approach allows for better accuracy of the instrumentation measurements (by minimizing the observer effect) and, thus, more effective analysis.

This process is not without cost. In particular, prior to creating instrumentation snapshots, deep instrumentation is enabled. This practice results in much more accurate analysis, but may lead to

## **A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information**

---

increased response times and CPU use during the data collection process. In addition, the analysis can be quite CPU-intensive.

For more detailed information about Adaptive Instrumentation, including step-by-step usage instructions, see the Precise for J2EE online help.

### **The Instrumentation Explorer tool**

The Instrumentation Explorer (IXP) is an excellent tool for seeing what is instrumented within your JVM. It can be particularly helpful in finding missing instrumentation or instrumentation that should be ignored. The IXP does not directly affect performance itself but, instead, provides valuable information for making other configuration choices. Using the IXP in this manner is most effective if the JVM is running in expert mode (see instrumentation modes). Here are some example use cases for the IXP:

- You wonder about the instrumentation status of a particular method or class. Open the IXP search dialog, and enter the package/method name you are looking for. If the method is not found, it might be ignored by the default instrumentation. If the method is listed but not enabled, you can add it using custom instrumentation or by selecting the checkbox next to the method and clicking the “Apply Changes” button.
- You open the IXP and sort by invocation count so that methods with zero or very few invocations are listed. These methods/classes could be good candidates for adding to the ignore list since they are rarely, if ever, used.
- You open the IXP and sort by invocation count so that methods with very large numbers of invocations are listed. These methods/classes could be good candidates for instrumenting (if not already instrumented) using custom instrumentation or the IXP UI.
- You open the IXP and sort by the “changed by” column. You can now see the methods instrumented by default, IXP, or Adaptive Instrumentation. You can also see methods disabled by IXP, Adaptive Instrumentation, or Over Instrumentation Protection.

For more detailed information about Instrumentation Explorer, including step-by-step use instructions, see the Precise for J2EE online help.

### **Application server metrics**

Precise for J2EE includes a collection of Java Management Extension (JMX) application server metrics that can have a significant impact on your application in terms of additional CPU and memory use. The default list of metrics collected for any specific application server type has been selected to provide a good deal of information about an application server. However, collecting these metrics can be a resource-expensive operation within an application server—and, if you are not using the metrics on a continual basis, then this is a continual loss of performance for minimal

gain.

It is suggested that you review the list of metrics being collected by using the Statistics workspace within the Precise for J2EE UI. Identify any metrics that are not useful for you, and disable them using the Statistics Administration tool.

For more detailed information about the Statistics Administration tool, see the Precise for J2EE online help.

### Ignore the junk

Perhaps one of the more powerful ways to reduce instrumentation overhead is to define the Java packages/classes that are ignored by the Precise for J2EE instrumenter. As mentioned previously, the default configuration includes the definition of common packages and classes that are ignored during instrumentation. These include core Java packages, common open source utilities, and the core classes of the most common application servers. The reason for ignoring these is simple: There is limited value in instrumenting these classes, as you cannot typically do anything about performance problems related to their use.

You should take the time to understand the various packages being instrumented within your application and to evaluate whether they should be added to the list of ignored classes. Doing so could significantly improve the startup time, memory consumption, and runtime performance of your instrumented application.

Example: Your application includes the use of a third-party rules engine defined by the package `com.rulerunner`. This package is called extensively within your application, and internally it is very recursive. Instrumenting the `com.rulerunner` classes with Precise for J2EE could have a significant impact on the performance of the application. In this case, it might be wise to add the package `com.rulerunner` to the list of ignored classes and, instead, rely on the call to these packages from within your application to highlight performance issues.

---

#### Instructions for expert users

- The list of ignored packages for the entire server is defined in a file called `<J2EE_HOME>/config/instrumenter/Ignore.xml` found on every Precise for J2EE agent machine.
  - Adding new entries to this file, using the existing definitions as a guide, will cause additional code to be ignored.
  - Removing entries from this file will cause additional code to be instrumented.
  - To create a JVM-specific `Ignore.xml`, copy the file to the `config/<JVM_ID>` directory and update the appropriate reference within `<J2EE_HOME>/config/<JVM_ID>/InstrumenterConfigList.xml`.
-

## **Instrumentation definitions**

The default configuration includes the instrumentation definitions for the most common J2EE interfaces. At startup, each of the classes loaded is evaluated against each of these definitions, looking for a match. Depending on the size of the application, this process can take significant time. If a particular type of instrumentation is not applicable to your environment, then performance gains can be realized during startup by removing these types from the instrumentation definition.

**Example:** If your application does not use Enterprise JavaBeans™ (EJBs), then the sections pertaining to EJBs can be removed. Doing so will prevent this instrumentation definition from being loaded at startup, thereby improving startup time.

---

Instructions for expert users

- The list of instrumentation definitions is defined in a file unique for every JVM: `<J2EE_HOME>/config/<JVM_ID>/InstrumenterConfigList.xml`
  - By default, this file references definitions for various instrumentation types. Commenting out these references within the file will cause those types to be removed from the instrumentation list at JVM startup.
- 

## **Custom instrumentation definitions**

As stated above, the default configuration includes the instrumentation definitions for the most common J2EE interfaces. However, some of the code in your application may not match these interfaces. In this case, changing the instrumentation mode to Expert should instrument these methods. However, you may also want to fine-tune the instrumentation for these methods by adding your packages to the custom instrumentation configuration.

Doing so will see that your application code is instrumented, but it will also increase the performance impact on your JVM. By adding items to the custom instrumentation list, you force Precise for J2EE to instrument them. This is valuable, as you will gather more data for all your application classes—but its value comes at the cost of performance penalties ensuing from the additional instrumentation. The scope of these impacts will be determined by the number of classes that match the custom definition and by your overall use of those classes. Take care when adding items to the custom configuration, and be sure to evaluate the impact of your changes in an iterative process.

The configuration can get quite complex. For example, you can indicate that you want a specific method call to be instrumented ONLY when called from a separate specific package.

**Example:** You know that the problem area of your code is the reports package, but the code in this package does not match the J2EE standard interfaces. You use the IXP and Adaptive

## **A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information**

---

Instrumentation tools, but you are not sure if all the calls to this package are being instrumented. You decide to add this package to the custom instrumentation list to make sure that it will be instrumented.

---

Instructions for expert users

- The list of custom packages for the entire server is defined in the file `<J2EE_HOME> /config/instrumenter/Custom.xml`, found on every Precise for J2EE agent machine. Adding new entries to this file, using the existing definitions as a guide, will cause additional code to be instrumented.
  - Removing entries from this file will cause the code defined by those entries to be evaluated by the default instrumentation rules.
  - To create a JVM-specific custom.xml file, copy the file to the `config/<JVM_ID>` directory and update the appropriate reference within `<J2EE_HOME>/config/<JVM_ID>/InstrumenterConfigList.xml`.
-

## **Conclusion**

As was shown in the previous sections, the instrumentation choices you make when monitoring the performance of a JVM have a direct impact on the performance itself. In this respect, the instrumentation—and the data it provides—is never free: You always pay for it in terms of impact on startup time, response time, memory, or CPU use.

The default configuration of the instrumentation for a given JVM has been chosen to balance performance and data, but this configuration is not tuned to your application. If you approach the proper instrumentation of your application as an iterative process, during which you evaluate the changes to be made and monitor the corresponding impacts, you can determine the instrumentation configuration that best meets your monitoring and data needs. By using the tools available within Precise for J2EE including Adaptive Instrumentation, Instrumentation Explorer, and custom instrumentation definitions, you can make this process both rapid and pain-free.

## **A Practical Approach to Balancing Application Performance and J2EE Instrumentation Information**

---

### **About Precise Software Solutions**

#### **Commitment, Focus, Experience**

For over 15 years Precise Software Solutions has helped our Global 2000 customers manage business performance in complex, heterogeneous environments, assuring availability and business continuity. Precise offers a complete solution – from discovery through ongoing management – that allows our customers to focus on their core business. We offer the broadest platform support, in terms of enterprise application, operating system, database, and development environment coverage. Precise is the solution of choice for IT as an organization-wide standard for application management.

#### **Visit our Web site**

[www.precise.com](http://www.precise.com)

#### **To speak with a Product Specialist in the U.S.**

Call toll-free 1 (877) 845 1886. For specific country offices and contact numbers, please visit our Web site.